

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04M 11/00	A1	(11) International Publication Number: WO 96/32805 (43) International Publication Date: 17 October 1996 (17.10.96)
--	----	---

(21) International Application Number: PCT/US96/04835
(22) International Filing Date: 10 April 1996 (10.04.96)
(30) Priority Data: 08/419,199 10 April 1995 (10.04.95) US
(71) Applicant (for all designated States except US): CORPORATE
COMPUTER SYSTEMS, INC. [US/US]; Building #4, 670
North Beers Road, Holmdel, NJ 07733 (US).
(72) Inventor; and
(75) Inventor/Applicant (for US only): HINDERKS, Larry, W.
[US/US]; 37 Ladwood Drive, Holmdel, NJ 07733 (US).
(74) Agents: SMALL, Dean, D. et al.; McAndrews, Held & Malloy,
Ltd., Suite 3400, 500 West Madison, Chicago, IL 60661
(US).

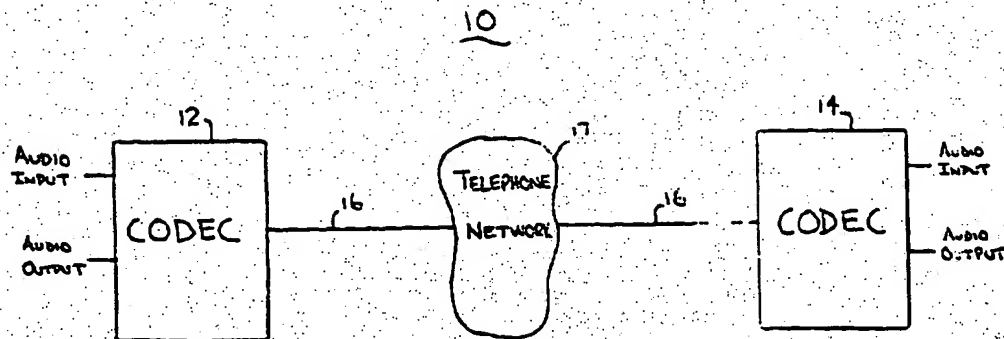
(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY,
CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS,
JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD,
MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD,
SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN,
ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent
(AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent
(AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU,
MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM,
GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.

BEST AVAILABLE COPY

(54) Title: METHOD AND APPARATUS FOR TRANSMITTING CODED AUDIO SIGNALS THROUGH A TRANSMISSION CHANNEL WITH LIMITED BANDWIDTH



(57) Abstract

A digital audio transmitter system (10) capable of transmitting high quality, wideband speech over a transmission channel with a limited bandwidth such as a traditional telephone line (16). The digital audio transmitter system (10) includes a coder (32) for coding an input audio signal to a digital signal having a transmission rate that does not exceed the maximum allowable transmission rate for traditional telephone lines and a decoder (40) for decoding the digital signal to provide an output audio signal with an audio bandwidth of wideband speech. A coder (32) and a decoder (40) may be provided in a single device (12) to allow two-way communication between multiple devices.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgyzstan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic	KR	Republic of Korea	SE	Sweden
CG	Congo	KZ	Kazakhstan	SG	Singapore
CH	Switzerland	LI	Liechtenstein	SI	Slovenia
CI	Côte d'Ivoire	LK	Sri Lanka	SK	Slovakia
CM	Cameroon	LR	Liberia	SN	Senegal
CN	China	LT	Lithuania	SZ	Swaziland
CS	Czechoslovakia	LU	Luxembourg	TD	Chad
CZ	Czech Republic	LV	Latvia	TG	Togo
DE	Germany	MC	Monaco	TJ	Tajikistan
DK	Denmark	MD	Republic of Moldova	TT	Trinidad and Tobago
EE	Estonia	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	UG	Uganda
FI	Finland	MN	Mongolia	US	United States of America
FR	France	MR	Mauritania	UZ	Uzbekistan
GA	Gabon			VN	Viet Nam

**METHOD AND APPARATUS FOR TRANSMITTING CODED AUDIO
SIGNALS THROUGH A TRANSMISSION CHANNEL
WITH LIMITED BANDWIDTH**

RELATED APPLICATION

5 The present application relates to co-pending PCT application PCT/US96/04974, filed April 10, 1996, entitled "System For Compression and Decompression of Audio Signals For Digital Transmission" by the same inventor and assigned to the Assignee of the present application. The co-pending PCT application noted above is incorporated by reference in its entirety along with any appendices and attachments thereto.

10 **FIELD OF THE INVENTION**

The present invention relates generally to an apparatus and method for transmitting audio signals and pertains, more specifically, to an apparatus and method for transmitting a high quality audio signal, such as wideband speech, through a transmission channel having a limited bandwidth or transmission rate.

15 **BACKGROUND OF THE INVENTION**

Human speech lies in the frequency range of approximately 7 Hz to 10 kHz. Because traditional telephone systems only provide for the transmission of analog audio signals in the range of about 300 Hz to 3400 Hz or a bandwidth of about 3 kHz (narrowband speech), certain characteristics of a speaker's voice are lost and the voice sounds somewhat muffled. A telephone system capable of transmitting an audio signal

- 2 -

approaching the quality of face-to-face speech requires a bandwidth of about 6 kHz (wideband speech).

5 Known digital transmission systems are capable of transmitting wideband speech audio signals. However, in order to produce an output audio signal of acceptable quality with a bandwidth of 6 kHz, these digital systems require a transmission channel with a transmission rate that exceeds the capacity of traditional telephone lines. A digital system transmits audio signals by
10 coding an input audio signal into a digital signal made up of a sequence of binary numbers or bits, transmitting the digital signal through a transmission channel, and decoding the digital signal to produce an output audio signal. During the coding process the digital signal is
15 reduced or compressed to minimize the necessary transmission rate of the signal. One known method for compressing wideband speech is disclosed in Recommendation G.722 (CCITT, 1988). A system using the compression method described in G.722 still requires a
20 transmission rate of at least 48 kbit/s to produce wideband speech of an acceptable quality.

Because the maximum transmission rate over traditional telephone lines is 28.8 kbit/s using the most advanced modem technology, alternative transmission
25 channels such as satellite or fiber optics would have to be used with an audio transmission system employing the data compression method disclosed in G.722. Use of these alternative transmission channels is both expensive and inconvenient due to their limited
30 availability. While fiber optic lines are available, traditional copper telephone lines now account for an overwhelming majority of existing lines and it is unlikely that this balance will change anytime in the near future. A digital phone system capable of
35 transmitting wideband speech over existing transmission rate limited telephone phone lines is therefore highly desirable.

SUBSTITUTE SHEET (RULE 26)

OBJECTS OF THE INVENTION

The disclosed invention has various embodiments that achieve one or more of the following features or objects:

5 An object of the present invention is to provide for the transmission of high quality wideband speech over existing telephone networks.

10 A further object of the present invention is to provide for the transmission of high quality audio signals in the range of 20 Hz to at least 5,500 Hz over existing telephone networks.

15 A still further object of the present invention is to accomplish data compression on wideband speech signals to produce a transmission rate of 28.8 kbit/s or less without significant loss of audio quality.

20 A still further object of the present invention is to provide a device which allows a user to transmit and receive high quality wideband speech and audio over existing telephone networks.

25 A still further object of the present invention is to provide a portable device which is convenient to use and allows ease of connection to existing telephone networks.

30 A still further object of the present invention is to provide a device which is economical to manufacture.

35 A still further object of the present invention is to provide easy and flexible programmability.

SUMMARY OF THE INVENTION

30 In accordance with the present invention, the disadvantages of the prior art have been overcome by providing a digital audio transmitter system capable of transmitting high quality, wideband speech over a transmission channel with a limited bandwidth such as a traditional telephone line.

35 More particularly, the digital audio transmitter system of the present invention includes a coder for

coding an input audio signal to a digital signal having a transmission rate that does not exceed the maximum allowable transmission rate for traditional telephone lines and a decoder for decoding the digital signal to provide an output audio signal with an audio bandwidth of wideband speech. A coder and a decoder may be provided in a single device to allow two-way communication between multiple devices. A device containing a coder and a decoder is commonly referred to as a CODEC (COder/DECoder).

These and other objects, advantages and novel features of the present invention, as well as details of an illustrative embodiment thereof, will be more fully understood from the following description and from the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a digital audio transmission system including a first CODEC and second CODEC in accordance with the present invention.

Fig. 2 is a block diagram of a CODEC of Fig. 1.

Fig. 3 is a block diagram of an audio input/output circuit of a CODEC.

Fig. 4 is a detailed circuit diagram of the audio input portion of Fig. 3.

Fig. 5 is a detailed circuit diagram of the level LED's portion of Fig. 3.

Fig. 6 is a detailed circuit diagram of the headphone amp portion of Fig. 3.

Fig. 7 is a block diagram of a control processor of a CODEC.

Fig. 8 is a detailed circuit diagram of the microprocessor portion of the control processor of Fig. 7.

Fig. 9 is a detailed circuit diagram of the memory portion of the control processor of Fig. 7.

Fig. 10 is a detailed circuit diagram of the dual UART portion of the control processor of Fig. 7.

Fig. 11 is a detailed circuit diagram of the keypad, LCD display and interface portions of the control processor of Fig. 7.

Fig. 12 is a block diagram of an encoder of a CODEC.

Fig. 13 is a detailed circuit diagram of the encoder digital signal processor and memory portions of the encoder of Fig. 12. Fig. 14 is a detailed circuit diagram of the clock generator portion of the encoder of Fig. 12.

Fig. 15 is a detailed circuit diagram of the Reed-Soloman encoder and decoder portions of Figs. 12 and 16.

Fig. 16 is a block diagram of a decoder of a CODEC.

Fig. 17 is a detailed circuit diagram of the encoder digital signal processor and memory portions of the decoder of Fig. 16.

Fig. 18 is a detailed circuit diagram of the clock generator portion of the decoder of Fig. 16.

Fig. 19 is a detailed circuit diagram of the analog/digital converter portion of the encoder of Fig. 12.

Fig. 20 is a detailed circuit diagram of the digital/analog converter portion of the decoder of Fig. 16.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A digital audio transmission system 10, as shown in Fig. 1, includes a first CODEC (COder/DECoder) 12 for transmitting and receiving a wideband audio signal such as wideband speech to and from a second CODEC 14 via a traditional copper telephone line 16 and telephone network 17. When transmitting an audio signal, the first CODEC 12 performs a coding process on the input analog audio signal which includes converting the input audio signal to a digital signal and compressing the

digital signal to a transmission rate of 28.8 kbit/s or less. The preferred embodiment compresses the digital using a modified version of the ISO/MPEG (International Standards Organization/Motion Picture Expert Groups) compression scheme according to the software routine disclosed in the microfiche software appendix filed herewith. The coded digital signal is sent using standard modem technology via the telephone line 16 and telephone network 17 to the second CODEC 14. The second CODEC 14 performs a decoding process on the coded digital signal by correcting transmission errors, decompressing the digital signal and reconverting it to produce an output analog audio signal.

Fig. 2 shows a CODEC 12 which includes an analog mixer 20 for receiving, amplifying, and mixing an input audio signal through a number of input lines. The input lines may include a MIC line 22 for receiving an analog audio signal from a microphone and a generic LINE 24 input for receiving an analog audio signal from an audio playback device such as a tape deck. The voltage level of an input audio signal on either the MIC line 22 or the generic LINE 24 can be adjusted by a user of the CODEC 12 by adjusting the volume controls 26 and 28. When the analog mixer 20 is receiving an input signal through both the MIC line 22 and the generic LINE 24, the two signals will be mixed or combined to produce a single analog signal. Audio level LED's 30 respond to the voltage level of a mixed audio signal to indicate when the voltage exceeds a desired threshold level. A more detailed description of the analog mixer 20 and audio level LED's 30 appears below with respect to Figs. 3 and 4.

The combined analog signal from the analog mixer 20 is sent to the encoder 32 where the analog signal is first converted to a digital signal. The sampling rate used for the analog to digital conversion is preferably one-half the transmission rate of the signal which will

ultimately be transmitted to the second CODEC 14 (shown in Fig. 1). After analog to digital conversion, the digital signal is then compressed using a modified version of the ISO/MPEG algorithm. The ISO/MPEG compression algorithm is modified to produce a transmission rate of 28.8 kbit/s. This is accomplished by the software routine that is disclosed in the software appendix.

The compressed digital signal from the encoder 32 is then sent to an error protection processor 34 where additional error protection data is added to the digital signal. A Reed-Solomon error protection format is used by the error protection processor 34 to provide both burst and random error protection. The error protection processor 34 is described below in greater detail with respect to Figs. 12 and 15.

The compressed and error protected digital signal is then sent to an analog modem 36 where the digital signal is converted back to an analog signal for transmitting. As shown in Fig. 1, this analog signal is sent via a standard copper telephone line 16 through a telephone network 17 to the second CODEC 14. The analog modem 36 is preferably a V.34 synchronous modem. This type of modem is commercially available.

The analog modem 36 is also adapted to receive an incoming analog signal from the second CODEC 14 (or another CODEC) and reconvert the analog signal to a digital signal. This digital signal is then sent to an error correction processor 38 where error correction according to a Reed-Soloman format is performed.

The corrected digital signal is then sent to a decoder 40 where it is decompressed using the modified version of the ISO/MPEG algorithm as disclosed in the software appendix. After decompression the digital signal is converted to an analog audio signal. A more detailed description of the decoder 40 appears below with respect to Figs. 7, 16, 17 and 18. The analog

audio signal may then be perceived by a user of the CODEC 12 by routing the analog audio signal through a headphone amp 42 wherein the signal is amplified. The volume of the audio signal at the headphone output line 44 is controlled by volume control 46.

The CODEC 12 includes a control processor 48 for controlling the various functions of the CODEC 12 according to software routines stored in memory 50. A more detailed description of the structure of the control processor appears below with respect to Figs. 7, 8, 9, 10, and 11. One software routine executed by the control processor allows the user of the CODEC 12 to initiate calls and enter data such as phone numbers. When a call is initiated the control processor sends a signal including the phone number to be dialed to the analog modem 36. Data entry is accomplished via a keypad 52 and the entered data may be monitored by observation of an LCD 54. The keypad 52 also includes keys for selecting various modes of operation of the CODEC 12. For example, a user may select a test mode wherein the control processor 48 controls the signal path of the output of the encoder to input of decoder to bypass the telephone network allows testing of compression and decompression algorithms and their related hardware. Also stored in memory 50 is the compression algorithm executed by the encoder 32 and the decompression algorithm executed by the decoder 40.

Additional LED's 56 are controlled by the control processor 48 and may indicate to the user information such as "bit synchronization" (achieved by the decoder) or "power on". An external battery pack 58 is connected to the CODEC 12 for supplying power.

Fig. 3 shows a lower level block diagram of the analog mixer 20, audio level LED's 30 and analog headphone amp 42 as shown in Fig. 2. Figs. 4, 5 and 6 are the detailed circuit diagrams corresponding to Fig. 3.

- 9 -

Referring to Fig. 3 and 4, line input 210 is an incoming line level input signal while mic input 220 is the microphone level input. These signals are amplified by a line amp 300 and a mic amp 302 respectively and their levels are adjusted by line level control 304 and mic level control 306 respectively. The microphone and line level inputs are fed to the input mixer 308 where they are mixed and the resulting combined audio input signal 310 is developed.

Referring now to Figs. 3 and 5, the audio input signal 310 is sent to the normal and overload signal detectors, 312 and 314 respectively, where their level is compared to a normal threshold 316 which defines a normal volume level and a clip threshold 318 which defines an overload volume level. When the audio input signal 310 is at a normal volume level a NORM LED 320 is lighted. When the audio input signal 310 is at an overload volume level a CLIP LED 322 is lighted.

Referring now to Figs. 3 and 6, the audio input signal 310 is fed into the record monitor level control 324, where its level is adjusted before being mixed with the audio output signal 336 from the digital/analog converter 442 (shown in Fig. 16 and 20). The audio output signal 336 is fed to the local monitor level control 326 before it is fed into the headphone mixer amplifier 334. The resulting output signal from the headphone mixer amplifier 334 goes to a headphone output connector 338 on the exterior of the CODEC 12 where a pair of headphones may be connected.

The audio input signal 310 and audio output signal 336 are fed to record mix control 328 which is operable by the user. The output of this control is fed to a mix level control 330 (also operable by a user) and then to the record output amplifier 332. The resulting output signal of the record output amplifier 332 goes to a record output 340 on the exterior of the CODEC 12.

Fig. 7 shows a lower level block diagram of the control processor 48 (shown in Fig. 2). The encoder 406 (referenced as number 32 in Fig. 2) is further described in Fig. 12 while the decoder 416 (referenced as number 40 in Fig. 2) is refined in Fig. 16. Figs. 8, 9, 10, 11, 13, 14, 15, 17, 18, 19 and 20 are detailed circuit diagrams.

Referring to Figs. 7 and 8 the microprocessor 400 is responsible for the communication between the user, via keypad 412 and LCD display 414, and the CODEC 12. The keypad 412 is used to input commands to the system while the LCD display 414, is used to display the responses of the keypad 412 commands as well as alert messages generated by the CODEC 12.

Referring now to Figs. 7 and 9, the RAM (random access memory) 402 is used to hold a portion of the control processor control software routines. The flash ROM (read only memory) 404 holds the software routine (disclosed in the software appendix) which controls the modified ISO/MPEG compression scheme performed by encoder DSP 406 and the modified ISO/MPEG decompression scheme performed by the decoder DSP 416, as well as the remainder of the control processor control software routines.

Referring now to Figs. 7 and 10, the dual UART (universal asynchronous receiver/transmitter) 408 is used to provide asynchronous input/output for the control processor 48. The rear panel remote control port 409 and the rear panel RS232 port 411 are used to allow control by an external computer. This external control can be used in conjunction with or instead of the keypad 412 and/or LCD display 414.

Referring now to Figs. 7 and 11, the programmable interval timer circuit 410 is used to interface the control processor with the keypad and LCD display.

Referring now to Figs. 7, 8 and 13, the encoder DSP (digital signal processor) 434 receives a digital pulse

code modulated signal 430 from the analog/digital converter 450. The encoder DSP 434 performs the modified ISO/MPEG compression scheme according to the software routine (described in the software appendix) stored in RAM memory 436 to produce a digital output 418.

The A/D clock generation unit 439 is shown in Figs. 12 and 14. The function of this circuitry is to provide all the necessary timing signals for the analog digital converter 450 and the encoder DSP 434.

The Reed-Soloman error correction encoding circuitry 438 is shown in Figs. 12 and 15. The function of this unit is to add parity information to be used by the Reed-Soloman decoder 446 (also shown in Fig. 16) to repair any corrupted bits received by the Reed-Soloman decoder 446. The Reed-Soloman corrector 438 utilizes a shortened Reed-Soloman GF(256) code which might contain, for example, code blocks containing 170 eight-bit data words and 8 eight-bit parity words.

Referring now to Figs. 7, 16 and 17, the decoder DSP 440 receives a digital input signal 422 from the modem 36 (shown in Fig. 2). The decoder DSP 440 performs the modified ISO/MPEG decompression scheme according to the software routine (described in the software appendix) stored in RAM memory 444 to produce a digital output to be sent to the digital/analog converter 442.

The D/A clock generation unit 448 is shown in Figs. 16 and 18. The function of this circuitry is to provide all the necessary timing signals for the digital/analog converter 442 and the decoder DSP 440.

The analog/digital converter 450, shown in Figs. 12 and 19, is used to convert the analog input signal 310 into a PCM digital signal 430.

The digital/analog converter 442, shown in Figs. 16 and 20 is used to convert the PCM digital signal from

the decoder DSP 440 into an analog audio output signal 336.

5 The Reed-Soloman error correction decoding circuitry 446, shown in Figs. 15 and 16, decodes a Reed-Soloman coded signal to correct errors produced during transmission of the signal through the modem 36 (shown in Fig. 2) and telephone network.

10 Another function contemplated by this invention is to allow real time, user operated adjustment of a number of psycho-acoustic parameters of the ISO/MPEG compression/decompression scheme used by the CODEC 12. A manner of implementing this function is described in applicant's application entitled "System For Adjusting Psycho-Acoustic Parameters In A Digital Audio Codec" 15 which is being filed concurrently herewith (such application and related Software Appendix are hereby incorporated by reference). Also, applicants application entitled "System For Compression And Decompression Of Audio Signals For Digital Transmission" 20 and related Software Appendix which are being filed concurrently herewith are hereby incorporated by reference.

25 This invention has been described above with reference to a preferred embodiment. Modifications and variations may become apparent to one skilled in the art upon reading and understanding this specification. It is intended to include all such modifications and alterations within the scope of the appended claims.

- 13 -

(c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.

nolist

;\DGCST\def.asm

This file contains the definitions for various structures.

The following is the minimum value for slb. The true value is -1 but that causes some computational difficulties so -120 db is used. The minimum value (2**-23) is about -138 db so there is some room left below -120 db

```
define MINDB      '-6228589'      ; -120 dB in slb's
define MINDB      '-.73'          ; -120 dB in slb's
```

Define the IO for the watch dog timer for bit set and bit clears

```
define WATCH_DOG      '#7.x:<<SFFE4' ; M_PBD bit 7 watch dog timer
```

The following defines the sampling rates

```
define SAM32K      '0'      ; sampling rate of 32 kHz
define SAM48K      '1'      ; sampling rate of 48 kHz
!!!28.8
define SAM16K      '2'      ; sampling rate of 14.4 kHz
define SAM24K      '3'      ; sampling rate of 14.4 kHz
define SAM16K      '2'      ; sampling rate of 16 kHz
define SAM24K      '3'      ; sampling rate of 24 kHz
!!!28.8
define SAM441K      '4'      ; sampling rate of 44.1 kHz
!!!28.8
define SAMTYPE      '2'      ; set the sampling rate to 14.4 kHz
!!!28.8
```

The following defines various parameters

```
define Numpfft      '1024'      ; number of points used by the fft
```

The following define the types of maskers.
ENDMSKR is not counted in the nmaskers count.

```
define DELETEDMSKR  '0'      ; the masker type of deleted
define NONTONAL     '1'      ; the masker type of non-tonal
define TONAL        '2'      ; the masker type of tonal
define ENDMSKR      '3'      ; the last masker in the array
```

The following define a tonal structure.

This structure occupies both x and y memory (1).

```
define TONALSSIZE    '2'      ; length of the structure
define TONALSPWRDB   '0'      ; offset to the tonal power (1)
define TONALSBIN     '1'      ; offset to the bin (x)
define MAXTONALS     '50'     ; the maximum number of tonals
```

The following define the sync info for the receiver. The sync pattern may be in general any NSYNC bits. The SYNCMSK must contain NSYNC 1's right justified and is used to isolate the sync word. MUSICAM uses 12 1's as



- 14 -

; the sync word.

```

define SYNC          'S000fff'      ;sync pattern left justified
define SYNCMSK       'S000fff'      ;mask high order from getvalue
define NSYNC         '12'           ;len sync word (hdr bits 0-11)

```

; For framing purposes by the decoder and unpadded frames, 24 bits are used:

```

; the 1st 12 bits must be 1's
; the next 4 bits are the 1st 4 bits of frame header of
;   the constant 'C' (1100);
; skip over the next 4 bits of the frame header that are reserved
;   for the bit rate
; the next 2 bits (01) of the frame header that represent sampling rate:
; '01' = 48 K sampling rate
; '10' = 32 K sampling rate

```

```

!!!28.8
; '00' = 24 K sampling rate (14.4 K rate)
; '00' = 16 K sampling rate (14.4 K rate)
; '11' = 24 K sampling rate
; '00' = 16 K sampling rate

```

```

!!!28.8
; the next 2 constant 0 bits of the frame header.
; The SYNCMSK must conform to the right justified framing sync pattern is used
; to isolate the sync word.

```

```

define FRAMESYNC_48K 'Sfffc04'      ;sync pattern for 48 K sampling
define FRAMESYNC_32K 'Sfffc08'      ;sync pattern for 32 K sampling
!!!28.8
define FRAMESYNC_24K 'Sfffc0c'      ;sync pattern for 24 K sampling
define FRAMESYNC_24K 'Sfffc00'      ;sync pattern - 14.4 K sampling
define FRAMESYNC_16K 'Sfffc00'      ;sync pattern - 14.4 K sampling
define FRAMESYNC_16K 'Sfffc00'      ;sync pattern for 16 K sampling
!!!28.8
define FRAMENSYNC    '24'           ;len sync word (hdr bits 0-23)
define FRAMESYNCSK   'Sffff0f'      ;mask reflect framing sync ptn
define GETSYNCSK     'S000fff'      ;mask high order from getvalue

```

```

; The following define the number of bits used by the fixed part of the
; MUSICAM frame.

```

```

define NSYST         '20'           ;length of the system info header
; define the use of protection check sum or not
define CRC_NO_PROTECT '0'           ; protection does not apply
define CRC_PROTECT    '1'           ; protection applies
define NCRCBITS       '16'           ; 16 bit check sum
define MASKCRC         'S00ffff'     ;mask high order from getvalue
define CRC_SUM_BIT_OFFSET '16'       ; 16th bit offset start at bit rate
; to calculate checksum
define CRC_VALUE       'S800500'     ; checksum divisor
define CRC_STORED_BIT_OFFSET '16'    ; bit offset to store checksum
; following the 32 bit header

```

```

; define the number of bits to be included in the checksum
; for the header and the checksum itself
; for one channel in mono

```

BAD ORIGINAL

SUBSTITUTE SHEET (RULE 26)

- 15 -

```

define CRC_BITS_A      '32'      ; incl bits from hdr & checksum
define CRC_BITS_B      '142'     ; incl bits per used channel
                                ; BALS = 88, SBits = 54

; code for the new ISO frame header (these are coded as left justified)

define SYSTHDR_1_NO_PROT      '000000d' ; bits 12-15: 1101 (4 bits)
define SYSTHDR_1_NO_PROT_LOW  '0000005' ; bits 12-15: 0101 (4 bits)
define SYSTHDR_1_PROTECT      '000000c' ; bits 12-15: 1100 (4 bits)
define SYSTHDR_1_PROTECT_LOW  '0000004' ; bits 12-15: 0100 (4 bits)

define SYSTHDR_2      '0000000' ; hdr bits 22-23: 00 (2 bits)

; use Copyright bit to indicate to decoder if CCS compression applies:
; bit 28: 0 means NO CCS compression
;         1 means audio coded with CCS compression

define SYSTHDR_3_NO_CCS_COMPRESS      '0000000' ; bits 28-31: 0000 (4)
define SYSTHDR_3_CCS_COMPRESS         '0000008' ; bits 28-31: 1000 (4)

define NSYSTHDR_1      '4'      ; 4 bits for header field 1
define NSYSTHDR_2      '2'      ; 2 bits for header field 2
define NSYSTHDR_3      '4'      ; 4 bits for header field 3
define MASKSYSTHDR_1    '000000f' ; mask high order from getvalue
define MASKSYSTHDR_2    '0000003' ; mask high order from getvalue
define MASKSYSTHDR_3    '000000f' ; mask high order from getvalue

; codes for the type of framing (2 bits in bits 24-25 of frame header)

define FULL_STEREO      '0000000' ; 00 stereo-left & right channels
define JOINT_STEREO     '0000001' ; 01 stereo intensity-2 channels
define DUAL              '0000002' ; 10 dual-2 channels
define MONO              '0000003' ; 11 mono-1 channel only

define NFRAMETYPE       '2'      ; 2 bits for type of frame field
define MASKFRAMETYPE    '0000003' ; mask high order from getvalue

; bit flags for controlling the type of framing during bit allocation & coding

define STEREO_vs_MONO    '0'      ; 0 = 2 channels, 1 = one
define LEFT_vs_RIGHT     '1'      ; 0 = left channel, 1 = right
define JOINT_FRAMING     '2'      ; 0 = not JOINT STEREO, 1 = yes
define JOINT_at_FULL      '3'      ; FULL Stereo upgrade allocation
                                ; 1 = YES at full, 0 = joint
define JOINT_at_SB_BOUND  '4'      ; has stereo intensity sub-band
                                ; boundary been reached:
                                ; 0 = NO, 1 = YES
define FIRST_TIME        '5'      ; did loop thru allocation tests
                                ; make any new bit allocation
                                ; 0 = yes, 1 = no
define MASKING_PASS      '6'      ; allocate to masking threshold:
                                ; 0=YES, 1=no (ALL are below)
define HEARING_PASS      '7'      ; alloc to threshold of hearing:
                                ; 0=YES, 1=no (ALL are below)
define FINAL_PASS        '8'      ; allocate pass of what's left:
                                ; 0 = NO, 1 = YES
define AT_LIMIT_SUBBAND   '9'      ; does NOT req at least 1 alloc
define AT_USED_SUBBAND    '10'     ; above used sub-band limit
define SUMMARY_ALARM     '16'     ; did any alarm get sensed
                                ; 0 = NO, 1 = YES

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

```

- 16 -

define PROTECT '18' ;should checksum (CRC16) protect
; 0 = NO, 1 = YES
define MONO_OUT_CHANNEL '19' ;output to only one channel:
; 0 = left, 1 = right
define MONO_OUT_BOTH '20' ;output mono to both channels:
; 0 = NO only one, 1 = YES
define LEFT_SINE_WAVE '21' ;left channel music vs tone
; 0 = NO only one, 1 = YES
define RIGHT_SINE_WAVE '22' ;right channel music vs tone
; 0 = NO only one, 1 = YES
define LOW_vs_HIGH_SAMPLING '23' ;encode low or high sample rate:
; 0 = low, 1 = high

;decoding overload flag

define SKF_ZERO '3' ;sensed a zero scale factor
; 0 = no, 1 = yes

;define bit position flags for decoding frames with the CRC-16 checksum

define USE_SAVED '6' ;checksum failed use saved frame
define FRAME_SAVED '7' ;a good frame was saved for use
define SAVE_FRAME '8' ;save this good frame for use
define USING_SAVED '9' ;this frame is the saved frame
define REFRAME '10' ;cnt bit errors exceeded, reframe

;define decoder auto selection flags for:
; bit rate (determined by trying to frame at each of the two
; bit rate choices)
; type of audio data (MUSICAM frames or G722)
; (determined by not being able to frame at either
; of the two bit rate choices)
; sampling rate (determined from a MUSICAM frame header)
; (if NOT auto selected, some other switch sets the value)

define AUTO_SELECT_BIT_RATE '11' ;0=NO, 1=YES
define AUTO_SELECT_DATA_TYPE '12' ;0=NO, 1=YES
define AUTO_SELECT_SAMPLE_RATE '13' ;0=NO, 1=YES
define MUSICAM_vs_G722 '14' ;0=MUSICAM, 1=G722
define SAMPLE_RATE_LOW_vs_HIGH '15' ;0=low, 1=high

; this flag indicates if CCS compression applies to getdata.asm

define DECOMPRESS_PACKED '16'

;this flag indicates that the framing process has previously determined
; that the input data to the MICRO decoder is a stream of MUSICAM frames

define MUSICAM_INPUT_SET '17' ;0=NO, 1=YES

;define flag that the current frame has a sync word violation

define NO_SYNC '21'

;define flag that determines which ISO CRC-16 controls to use:
; 0 = OLD controls: seed with 0's and fixed span of bits covered
; 1 = NEW controls: seed with F's and dynamic span over the SBits

define CRC_OLD_vs_NEW '22'

```

- 17 -

```

;define the sub-band allocation AtLimit bit flags that control selection
define MASKING_LIMIT '0' ;1 reached sub-band's masking threshold
define HEARING_LIMIT '1' ;1 reached sub-band's hearing threshold
define ALLOCATE_LIMIT '2' ;1 reached sub-band's max bit limit
define NO_ALLOCATE '3' ;1 NO allocation at this sub-band

;define the standard limit of sub-bands requiring at least 1 level of
; allocation even if the signal is below the Global Masking Threshold
define LIMITSUBBANDS '17' ;sub-bands 0 thru 16 get at least 1

;define the number of successive frames that a sub-band did not need any bits
; allocated before shutting the sub-band from being allocated
define FRAMELIMIT '4'

; codes for stereo intensity subband bound (2 bits 25-27 of frame header)
define INTENSITY_4 '0000000' ; 00 subbands 4-31 intensity mode
define INTENSITY_8 '0000001' ; 01 subbands 8-31 intensity mode
define INTENSITY_12 '0000002' ; 10 subbands 12-31 intensity mode
define INTENSITY_16 '0000003' ; 11 subbands 16-31 intensity mode

define NSTINTENSITY '2' ; 2 bits for intensity boundary
define MASKSTINTENSITY '0000003' ; mask high order from getvalue

; stereo intensity boundary sub-band counts
define BOUND_4 '4' ; 0-3 full stereo, 4-31 intensity
define BOUND_8 '8' ; 0-7 full stereo, 8-31 intensity
define BOUND_12 '12' ; 0-11 full stereo, 12-31 intensity
define BOUND_16 '16' ; 0-15 full stereo, 16-31 intensity

; codes for output bit rates (4 bits in positions 16-19 of frame header)
define BITRATE_FREE '0000000' ; 0000 @ unknown kbits/s
define BITRATE_32 '0000001' ; 0001 @ 32 kbits/s
define BITRATE_48 '0000002' ; 0010 @ 48 kbits/s
;!!!28.8
define BITRATE_56 '0000003' ; 0011 @ 28.8 kbits/s
define BITRATE_64 '0000003' ; 0011 @ 28.8 kbits/s
;
define BITRATE_56 '0000003' ; 0011 @ 56 kbits/s
define BITRATE_64 '0000004' ; 0100 @ 64 kbits/s
;!!!28.8
define BITRATE_80 '0000005' ; 0101 @ 80 kbits/s
define BITRATE_96 '0000006' ; 0110 @ 96 kbits/s
define BITRATE_112 '0000007' ; 0111 @ 112 kbits/s
define BITRATE_128 '0000008' ; 1000 @ 128 kbits/s
define BITRATE_160 '0000009' ; 1001 @ 160 kbits/s
define BITRATE_192 '000000a' ; 1010 @ 192 kbits/s
define BITRATE_224 '000000b' ; 1011 @ 224 kbits/s
define BITRATE_256 '000000c' ; 1100 @ 256 kbits/s
define BITRATE_320 '000000d' ; 1101 @ 320 kbits/s
define BITRATE_384 '000000e' ; 1110 @ 384 kbits/s

; low sample rates: 24000, 22050 and 16000
; codes for output bit rates (4 bits in positions 16-19 of frame header)
define BITRATE_FREE_LOW '0000000' ; 0000 @ unknown kbits/s

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 18 -

```

define BITRATE_8_LOW      'S000001' ; 0001 @ 8 kbits/s
define BITRATE_16_LOW     'S000002' ; 0010 @ 16 kbits/s
define BITRATE_24_LOW     'S000003' ; 0011 @ 24 kbits/s
define BITRATE_32_LOW     'S000004' ; 0100 @ 32 kbits/s
define BITRATE_40_LOW     'S000005' ; 0101 @ 40 kbits/s
define BITRATE_48_LOW     'S000006' ; 0110 @ 48 kbits/s
define BITRATE_56_LOW     'S000007' ; 0111 @ 56 kbits/s
define BITRATE_64_LOW     'S000008' ; 1000 @ 64 kbits/s
define BITRATE_80_LOW     'S000009' ; 1001 @ 80 kbits/s
define BITRATE_96_LOW     'S00000a' ; 1010 @ 96 kbits/s
define BITRATE_112_LOW    'S00000b' ; 1011 @ 112 kbits/s
define BITRATE_128_LOW    'S00000c' ; 1100 @ 128 kbits/s
define BITRATE_144_LOW    'S00000d' ; 1101 @ 144 kbits/s
define BITRATE_160_LOW    'S00000e' ; 1110 @ 160 kbits/s

define NBITRATE           '4' ; 4 bits for bit rate code in hdr
define MASKNBITRATE       'S00000f' ; mask high order from getvalue

; codes for input sampling rate (2 bits in positions 20-21 of frame header)
!!!28.8
define SAMPLE_ID_BIT_HIGH '1'
define SAMPLINGRATE_16    'S000000' ; 00 @ 14.4 kHz
define SAMPLINGRATE_24    'S000000' ; 00 @ 14.4 kHz
define SAMPLINGRATE_16    'S000000' ; 00 @ 16 kHz
define SAMPLINGRATE_48    'S000001' ; 01 @ 48 kHz
define SAMPLINGRATE_32    'S000002' ; 10 @ 32 kHz
define SAMPLINGRATE_24    'S000003' ; 11 @ 24 kHz
!!!28.8

define NSAMPLERATE        '2' ; 2 bits for sampling rate in hdr
define MASKNSAMPLERATE    'S000003' ; mask high order from getvalue

define NSBITS             '2' ; length of the scale factor select
define MASKNSBITS         'S000003' ; mask high order from getvalue

; The following defines the masker structure.
; This structure occupies both x and y memory (1).

define MASKERSIZE         '3' ; length of the structure
define MASKERSPWRDB       '0' ; offset to masker power (1 for watts
; and x for dB)
define MASKERSRDPWRDB     '0' ; offset to reduced power in db (y)
define MASKERSBIN         '1' ; offset to bin number (x)
define MASKERSBFREQ       '1' ; offset to freq in bark (y)
define MASKERSTYPE        '2' ; offset to masker type (x)
define MASKERSCRITBND     '2' ; offset to masker crital band if noisy.

; highest number of critical bands for all sampling rates
define NUMMAXCRITBND     '26'

if SAMTYPE==SAM16K
!!!28.8
define MAXCRITBND        '21' ; number of critical bands
!!!28.8
endif

if SAMTYPE==SAM24K
!!!28.8

```


- 19 -

```

define MAXCRITBND5 '21' ;number of critical bands
define MAXCRITBND5 '23' ;number of critical bands
!!!28.8
endif

if SAMTYPE==SAM32K
define MAXCRITBND5 '24' ;number of critical bands
endif

if SAMTYPE==SAM48K
define MAXCRITBND5 '24' ;number of critical bands
endif

!!!28.8
define MAXCRITBND5_16 '21' ;number of critical bands at 14.4 K
define MAXCRITBND5_24 '21' ;number of critical bands at 14.4 K
define MAXCRITBND5_16 '21' ;number of critical bands at 16 K
define MAXCRITBND5_24 '23' ;number of critical bands at 24 K
!!!28.8
define MAXCRITBND5_32 '24' ;number of critical bands at 32 K
define MAXCRITBND5_48 '24' ;number of critical bands at 48 K

; The following defines the Aliasing structure
; This structure only occupies x or y memory

define ALIASIZE '2' ;length of the structure
define ALIASBIN '0' ;bin number of aliazer (0-511)
define ALIASPWRDB '1' ;power of the aliazer in s1b

; General things

define NUMSUBBANDS '32' ;number of sub-bands
define NUMBLOCKS '3' ;number of blocks per super-frame
define NUMPERBLK '384' ;number of points per block
define NUMPERSUBBAND '12' ;number of points per sub-band
define SKF '6' ;number of bits per scale factor
define MASKSKF '00003f' ;mask high order from getvalue
define SKFX2 '64' ;number of scale factors
define BINSBERSUBBAND '16' ;number of FFT bins per subband
define NUMCHANNELS '2' ;two channels: left and right
define NUMSNRPOSITIONS '18' ;18 Signal-to-Noise position codes
define NUMINDEXES '16' ;16 position codes Allowed per sub-band

define MAXSUBBANDS_CCS '30' ;maximum sub-bands to ever be used
define MINSUBBANDS_CCS '4' ;minimum sub-bands to ever be used
define MAXSUBBANDS_LO '14' ;low bit rate max sub-bands ever used

;define the used subbands for 64 and 56 KBits
(sampling rate / 2) = max Hz / by 32 sub-bands = Hz per sub-band
based on sampling rate:
14400 @ 225 Hz per sub-band (14400/(2*32:NUMSUBBANDS) = 225
16000 @ 250 Hz per sub-band (16000/(2*32:NUMSUBBANDS) = 250
24000 @ 375 Hz per sub-band (24000/(2*32:NUMSUBBANDS) = 375
32000 @ 500 Hz per sub-band (32000/(2*32:NUMSUBBANDS) = 500
48000 @ 750 Hz per sub-band (48000/(2*32:NUMSUBBANDS) = 750
also based on bandwidth code selection from a pair external switches:
00 = CCS standard
01 = 1 sub-band less than standard
10 = 2 sub-bands less than standard

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 20 -

11 - 3 sub-bands less than standard

```

define USEDSUBBANDS_00_16 '27' ; 6750 Hz @ 16000 Hz sampling
define USEDSUBBANDS_01_16 '26' ; 6500 Hz @ 16000 Hz sampling
define USEDSUBBANDS_10_16 '25' ; 6250 Hz @ 16000 Hz sampling
define USEDSUBBANDS_11_16 '24' ; 6000 Hz @ 16000 Hz sampling
;!!!28.8
define USEDSUBBANDS_00_16 '30' ; 6750 Hz @ 14400 Hz sampling
define USEDSUBBANDS_01_16 '26' ; 5850 Hz @ 14400 Hz sampling
define USEDSUBBANDS_10_16 '22' ; 4950 Hz @ 14400 Hz sampling
define USEDSUBBANDS_11_16 '18' ; 4050 Hz @ 14400 Hz sampling
define USEDSUBBANDS_00_16 '22' ; 5500 Hz @ 16000 Hz sampling
define USEDSUBBANDS_01_16 '21' ; 5250 Hz @ 16000 Hz sampling
define USEDSUBBANDS_10_16 '20' ; 5000 Hz @ 16000 Hz sampling
define USEDSUBBANDS_11_16 '18' ; 4500 Hz @ 16000 Hz sampling
;!!!28.8
;!!!28.8
define USEDSUBBANDS_00_24 '30' ; 6750 Hz @ 14400 Hz sampling
define USEDSUBBANDS_01_24 '26' ; 5850 Hz @ 14400 Hz sampling
define USEDSUBBANDS_10_24 '22' ; 4950 Hz @ 14400 Hz sampling
define USEDSUBBANDS_11_24 '18' ; 4050 Hz @ 14400 Hz sampling
define USEDSUBBANDS_00_24 '27' ; 10125 Hz @ 24000 Hz sampling
define USEDSUBBANDS_01_24 '26' ; 9750 Hz @ 24000 Hz sampling
define USEDSUBBANDS_10_24 '25' ; 9375 Hz @ 24000 Hz sampling
define USEDSUBBANDS_11_24 '24' ; 9000 Hz @ 24000 Hz sampling
;!!!28.8
define USEDSUBBANDS_00_24 '18' ; 6750 Hz @ 24000 Hz sampling
define USEDSUBBANDS_01_24 '16' ; 6000 Hz @ 24000 Hz sampling
define USEDSUBBANDS_10_24 '14' ; 5250 Hz @ 24000 Hz sampling
define USEDSUBBANDS_11_24 '12' ; 4500 Hz @ 24000 Hz sampling

define USEDSUBBANDS_00_32 '20' ; 10000 Hz @ 32000 Hz sampling
define USEDSUBBANDS_01_32 '19' ; 9500 Hz @ 32000 Hz sampling
define USEDSUBBANDS_10_32 '18' ; 9000 Hz @ 32000 Hz sampling
define USEDSUBBANDS_11_32 '17' ; 8500 Hz @ 32000 Hz sampling

define USEDSUBBANDS_00_48 '11' ; 8250 Hz @ 48000 Hz sampling
define USEDSUBBANDS_01_48 '10' ; 7500 Hz @ 48000 Hz sampling
define USEDSUBBANDS_10_48 '9' ; 6750 Hz @ 48000 Hz sampling
define USEDSUBBANDS_11_48 '8' ; 6000 Hz @ 48000 Hz sampling

define INPCM '1152' ;NUMBERBLK*NUMBLOCKS
define PCMSIZE '2560' ;NUMBERBLK*NUMBLOCKS*2+256
define PCMSIZE '1152' ;NUMBERBLK*NUMBLOCKS !!!dbg!!!
define PCMSIZE '2304' ;NUMBERBLK*NUMBLOCKS*2 !!!dbg!!!

if SAMTYPE==SAM16K
;!!!28.8
define RATE56 '0' ;dip switch code for 28.8 Kbits
define OUTM56 '96' ;96 output words (2304 bits)
define OUTB56 '2304' ;.080 * 28800

define RATE64 '0' ;dip switch code for 28.8 Kbits
define OUTM64 '96' ;96 output words (2304 bits)
define OUTB64 '2304' ;.080 * 28800

define RATE56 '0' ;dip switch code for 56 Kbits
define OUTM56 '168' ;168k output words (4032 bits)
define OUTB56 '4032' ;.072 * 56000

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-21-

```

define RATE64      '1'      ;dip switch code for 64 Kbits
define OUTM64      '192'    ;192k output words (4608 bits)
define OUTB64      '4608'   ;.072 * 64000
;!!!28.8
endif

if SAMTYPE==SAM24K
;!!!28.8
define RATE56      '0'      ;dip switch code for 28.8 Kbits
define OUTM56      '96'     ;96 output words (2304 bits)
define OUTB56      '2304'   ;.080 * 28800

define RATE64      '0'      ;dip switch code for 28.8 Kbits
define OUTM64      '96'     ;96 output words (2304 bits)
define OUTB64      '2304'   ;.080 * 28800

define RATE56      '0'      ;dip switch code for 56 Kbits
define OUTM56      '112'    ;112k output words (2688 bits)
define OUTB56      '2688'   ;.048 * 56000

define RATE64      '1'      ;dip switch code for 64 Kbits
define OUTM64      '128'    ;128k output words (3072 bits)
define OUTB64      '3072'   ;.048 * 64000
;!!!28.8
endif

if SAMTYPE==SAM32K
define RATE56      '0'      ;dip switch code for 56 Kbits
define OUTM56      '84'     ;84k output words (2016 bits)
define OUTB56      '2016'   ;.036 * 56000

define RATE64      '1'      ;dip switch code for 64 Kbits
define OUTM64      '96'     ;96k output words (2304 bits)
define OUTB64      '2304'   ;.036 * 64000
endif

if SAMTYPE==SAM48K
define RATE56      '0'      ;dip switch code for 56 Kbits
define OUTM56      '56'     ;56k output words (1344 bits)
define OUTB56      '1344'   ;.024 * 64000

define RATE64      '1'      ;dip switch code for 64 Kbits
define OUTM64      '64'     ;64k output words (1536 bits)
define OUTB64      '1536'   ;.024 * 64000
endif

define RATE_LO      '0'      ;dip switch code for lower Kbit rate
define RATE_HI      '1'      ;dip switch code for higher Kbit rate

define framing bit rate values for sampling at 16 K

define OUTM32_16    '96'     ;96k output words (2304 bits)
define OUTB32_16    '2304'   ;.072 * 32000
define OUTM48_16    '144'    ;144k output words (3456 bits)
define OUTB48_16    '3456'   ;.072 * 48000
;!!!28.8
define OUTM56_16    '96'     ;96 output words (2304 bits)
define OUTB56_16    '2304'   ;.080 * 28800
define OUTM64_16    '96'     ;96 output words (2304 bits)

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 22 -

```

define OUTB64_16      '2304'      ;.080 * 28800
define OUTM56_16      '168'       ;168k output words (4032 bits)
define OUTB56_16      '4032'      ;.072 * 56000
define OUTM64_16      '192'       ;192k output words (4608 bits)
define OUTB64_16      '4608'      ;.072 * 64000
!!!28.8

; define framing bit rate values for sampling at 24 K

define OUTM32_24      '64'        ;64k output words (1536 bits)
define OUTB32_24      '1536'      ;.048 * 32000
define OUTM48_24      '96'        ;96k output words (2304 bits)
define OUTB48_24      '2304'      ;.048 * 48000
!!!28.8

define OUTM56_24      '96'        ;96 output words (2304 bits)
define OUTB56_24      '2304'      ;.080 * 28800
define OUTM64_24      '96'        ;96 output words (2304 bits)
define OUTB64_24      '2304'      ;.080 * 28800
define OUTM56_24      '112'       ;112k output words (2688 bits)
define OUTB56_24      '2688'      ;.048 * 56000
define OUTM64_24      '128'       ;128k output words (3072 bits)
define OUTB64_24      '3072'      ;.048 * 64000
!!!28.8

; define framing bit rate values for sampling at 32 K

define OUTM32_32      '48'        ;48k output words (1152 bits)
define OUTB32_32      '1152'      ;.036 * 32000
define OUTM48_32      '72'        ;72k output words (1728 bits)
define OUTB48_32      '1728'      ;.036 * 48000
define OUTM56_32      '84'        ;84k output words (2016 bits)
define OUTB56_32      '2016'      ;.036 * 56000
define OUTM64_32      '96'        ;96k output words (2304 bits)
define OUTB64_32      '2304'      ;.036 * 64000

; define framing bit rate values for sampling at 48 K

define OUTM32_48      '32'        ;32k output words (768 bits)
define OUTB32_48      '768'       ;.024 * 32000
define OUTM48_48      '48'        ;48k output words (1152 bits)
define OUTB48_48      '1152'      ;.024 * 48000
define OUTM56_48      '56'        ;56k output words (1344 bits)
define OUTB56_48      '1344'      ;.024 * 64000
define OUTM64_48      '64'        ;64k output words (1536 bits)
define OUTB64_48      '1536'      ;.024 * 64000

; highest number of freqs used for coding for all sampling rates

define MAXNMSKFREQS    '132'

; number of freqs used for coding based on defined sampling rates

if SAMTYPE==SAM16K
!!!28.8
define NMSKFREQS      '132'      ; number of freqs used for coding
!!!28.8
endif

if SAMTYPE==SAM24K
!!!28.8

```

- 23 -

```

define NMSKFREQS      '132' ;number of freqs used for coding
;:::28.8
endif

if SAMTYPE==SAM32K
define NMSKFREQS      '132' ;number of freqs used for coding
endif

if SAMTYPE==SAM48K
define NMSKFREQS      '126' ;number of freqs used for coding
endif

;:::28.8
define NMSKFREQS_16   '132' ;num freqs used for coding at 14.4 K
define NMSKFREQS_24   '132' ;num freqs used for coding at 14.4 K
define NMSKFREQS_16   '132' ;num freqs used for coding at 16 K
define NMSKFREQS_24   '132' ;num freqs used for coding at 24 K
;:::28.8
define NMSKFREQS_32   '132' ;num freqs used for coding at 32 K
define NMSKFREQS_48   '126' ;num freqs used for coding at 48 K

; the following indicates if CCS compression for positions: 1, 2 and 3
;
define COMPRESS      '0' ;0 indicates no CCS compression
define COMPRESS      '1' ;1 indicates use CCS compression

; define uncompressed getdata() getvalue masks for unpack:
;   upack3, upack5 and upack9
;
define MASKUPACK3     '500001f' ; 5 bit getvalue retrieved
define MASKUPACK5     '500007f' ; 7 bit getvalue retrieved
define MASKUPACK9     '50003ff' ; 10 bit getvalue retrieved

; define CCS compress: getdata() getvalue masks for unpack:
;   upack3, upack5, upack8 and upack9
;
define MASKUPACK3X    '500000f' ; 4 bit getvalue retrieved
define MASKUPACK5X    '500003f' ; 6 bit getvalue retrieved
define MASKUPACK8X    '50000ff' ; 8 bit getvalue retrieved
define MASKUPACK9X    '50003ff' ; 10 bit getvalue retrieved

; needed by the decoder rdecode program

define NOOF           '5' ;number of out of frames
define NSBUFS         '4' ;number of sync buffers
define MAX_TRIES      '10' ;restart after framing tries

; needed by the decoder rsynth program

define OUTBUF         '512' ;size of the output buffer
define OUTBUF         '768' ;size of the output buffer
define OUTBUF         '1024' ;size of the output buffer
define OUTBUF         '1152' ;size of the output buffer

; needed by all

define NPERGROUP      '3' ;number of samples per processing grp

; This constant is used by xpsycho only to set to offset used to account
; for the phase locked loop (PLL) jitter.

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 24 -

```

define PLLOFFSET      '32'      number of samples of offset

define the methods of operation controlled by external switches
normal operation vs various diagnostic operations

define NORMAL_OPER    'S000000' 000 normal operation
define LEFT_1000hz    'S000001' 001 1000 Hz tone left, mute right
define RIGHT_1000hz   'S000002' 010 1000 Hz tone right, mute left
define BOTH_1000hz    'S000003' 011 1000 Hz tone to both channels
define MEMORY_TEST    'S000004' 100 perform memory tests
define LEFT_10000hz   'S000005' 101 10000 Hz tone left, mute right
define RIGHT_10000hz  'S000006' 110 10000 Hz tone right, mute left
define BOTH_10000hz   'S000007' 111 10000 Hz tone to both channels

define ancillary data baud rates and byte counts per frame time period (msecs)

define BAUD300        '0'        ;dip switch code for 300 baud
define BYTES300       '1'        ;1 byte (7.2 bits ==> 8 bits)
define M_SCCR300      '557d'     ;set clock for 300 baud rate

define BAUD1200       '1'        ;dip switch code for 1200 baud
define BYTES1200      '4'        ;4 bytes (28.8 bits ==> 32 bits)
define M_SCCR1200     '515f'     ;set clock for 1200 baud rate

define BAUD2400       '2'        ;dip switch code for 2400 baud
define BYTES2400      '8'        ;8 bytes (57.6 bits ==> 64 bits)
define M_SCCR2400     '5af'      ;set clock for 2400 baud rate

define BAUD3600       '3'        ;dip switch code for 3600 baud
define BYTES3600      '11'       ;11 bytes (86.4 bits ==> 88 bits)
define M_SCCR3600     '574'      ;set clock for 3600 baud rate

define BAUD4800       '4'        ;dip switch code for 4800 baud
define BYTES4800      '15'       ;15 bytes (115.2 bits ==> 120 bits)
define M_SCCR4800     '557'      ;set clock for 4800 baud rate

define BAUD7200       '5'        ;dip switch code for 7200 baud
define BYTES7200      '22'       ;22 bytes (172.8 bits ==> 176 bits)
define M_SCCR7200     '53a'      ;set clock for 7200 baud rate

define BAUD9600       '6'        ;dip switch code for 9600 baud
define BYTES9600      '29'       ;29 bytes (230.4 bits ==> 232 bits)
define M_SCCR9600     '52b'      ;set clock for 9600 baud rate

define BAUD19200      '7'        ;dip switch code for 19200 baud
define BYTES19200     '58'       ;58 bytes (460.8 bits ==> 464 bits)
define M_SCCR19200    '515'      ;set clock for 19200 baud rate

define BAUD38400      '8'        ;dip switch code for 38400 baud
define BYTES38400     '116'      ;116 bytes (921.6 bits ==> 928 bits)
define M_SCCR38400    '5a'       ;set clock for 38400 baud rate

define BAUD_KMART_DCD '8'        ;code forced by box cti
define BYTE_KMART_42187 '127'    ;127 bytes (1012.5 bits ==> 1016 bits)
define M_KMART_42187  '59'       ;set clock for 42187.5 baud rate

define M_SCR_CD       'S05'      ;enable re & rei for encoder
define M_SCR_DCD      'S12'      ;enable te & tai for decoder

```

.25.

```
define DATABUFLEN      '512'    ;ancillary data input buffer length
define BITSPERBYTE     '8'      ;ancillary data in 8-bit bytes
define BITSFORPADDING  '3'      ;framed bit count for pad byte count
```

list

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



no.1st

-26-

(c) 1995. Copyright: Corporate Computer Systems, Inc. All rights reserved.

\DGCST\box_ctl.asm

This file contains the definitions for the control variables for running the encoder and decoder for:

Digicast MiniCodec version of CCS CDQ1000:
 sampling rate is 14.400 K - 225 Hz per sub-band (coded as 16 K sampling)
 bit rate is 28.8 KBits per sec (coded as the low sampling rate)
 the frame header is coded as 'fffc00'
 Port B for the encoder and decoder is defined as a host port
 encoder has its own phase lock detected on pc1 of Port C
 decoder phase lock is detected on pc0 of Port C
 ancillary data is NOT APPLICABLE

define the bits required for Reed Solomon error correction

define REED_SOLOMON_BITS '240' ;8 bits - 30 Reed Solomon bytes

define the choice pairs of input PCM sampling rates to make available

```

;!!!28.8
define SAMPLE_16K_AND_24K '0' ;choice of 14400 or 14400
define SAMPLE_16K_AND_24K '0' ;choice of 16000 or 24000
;!!!28.8
define SAMPLE_16K_AND_32K '1' ;choice of 16000 or 32000
define SAMPLE_16K_AND_48K '2' ;choice of 16000 or 48000
define SAMPLE_24K_AND_32K '3' ;choice of 24000 or 32000
define SAMPLE_24K_AND_48K '4' ;choice of 24000 or 48000
define SAMPLE_32K_AND_48K '5' ;choice of 32000 or 48000

```

define the selected pair of input PCM sampling rates to make available

```

;!!!28.6
define SAMPLE_RATE_PAIR '0' ;14400 and 14400 sample rates
;!!!28.8

```

```

;!!!28.8 if SAMPLE_RATE_PAIR==SAMPLE_16K_AND_24K
;!!!28.8

```

```

define LOW_SAMPLE_RATE 'S000000' ; 00 @ 14.4 KHz
define HIGH_SAMPLE_RATE 'S000000' ; 00 @ 14.4 KHz
define FRAMESYNC_LO 'Sfffc00' ; fr sync pattern 14.4K
define FRAMESYNC_HI 'Sfffc00' ; fr sync pattern 14.4K
define LOW_SAMPLE_RATE_CCS 'S000000' ; 00 @ 14.4(16) KHz
define HIGH_SAMPLE_RATE_CCS 'S000000' ; 00 @ 14.4(16) KHz
define FRAMESYNC_LO_CCS 'Sfffc00' ; fr sync old CCS 14.4K(16)
define FRAMESYNC_HI_CCS 'Sfffc00' ; fr sync old CCS 14.4K(16)
define LOW_SAMPLE_RATE_ISO 'S000000' ; 00 @ 14.4(16) KHz
define HIGH_SAMPLE_RATE_ISO 'S000000' ; 00 @ 14.4(24) KHz
define FRAMESYNC_LO_ISO 'Sfffc00' ; fr sync MPEG-ISO 14.4K(16)
define FRAMESYNC_HI_ISO 'Sfffc00' ; fr sync MPEG-ISO 14.4K(24)

```

```

;!!!28.8
;!!!28.8 endif

```

define the framing max tries for MUSICAM

define VERIFY_TRIES '5' ;verify found rates

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 27 -

```

define MAX_BOOT_TRIES      '40'      ;for 96 seconds
define MAX_AUTO_TRIES      '80'      ;for 1.92 seconds

;define the power up wait times before going into processing

define XCODE_STARTUP        '1000'    ;1 second
define RDCDSYNT_STARTUP     '1000'    ;1 second

;define the memory layouts for any diagnostic memory testing:

;decoder memory layout:

define START_P_MEMORY_DCD   '1024'
define END_P_MEMORY_DCD     '2048'
define START_X_MEMORY_DCD   '40'
define END_X_MEMORY_DCD     '5120'
define START_Y_MEMORY_DCD   '128'
define END_Y_MEMORY_DCD     '1536'

define WATCH_DOG_TEST_DCD   '20'      ;20 millisecs for watch dog

;define the encoder/decoder overload scale factor code a scale factor
; lower than this value is considered an overload condition

define OVERLOAD_SKF         '5'

;define the controls to reframe if an excessive error condition persists
; A frequency count of frames out-of-frame or oof's (no sync pattern)
; and a frequency count of checksum bit errors are maintained.
; For every bad frame condition the appropriate counter is incremented at
; a given value and for every good frame the counter is decremented at
; a lower value than it was incremented. A tolerance limit is tested against
; the counter when an error is sensed to see if it is time to force reframing.
; By decrementing at a lower rate would allow a counter to reach the reframe
; limit when there is a persistent pattern of alternating or nearly alternating
; good frames and bad frames.

define GOOD_DECREMENT       '1'      ;good frame decrement value
define BAD_INCREMENT        '2'      ;error condition frame increment value
define BAD_LIMIT            '4'      ;out-of-frame (oof's) tolerance
define BAD_CRC_LIMIT        '10'     ;CRC-16 checksum bit-error tolerance

;ben 3/8/94 (start): G722 modification for H221
; Hand shake definition (PBD)

define HSFTT                '#14'    ;PB14 input
define C0                    '#9'     ;PB9 input
define C2                    '#10'    ;PB10 input
define ABIT                  '#12'    ;PB12 input
define HSTTF                 '#13'    ;PB13 output

; Tx flag definition

define TX_FLAG               '#0'     ;#0 bit of x:flag
define M64                   '#1'     ; (PB1) M64 or M56 switch

;ben 3/8/94 (end): G722 modification for H221
;ben 3/21/95: decoder Reed Solomon address parameters

```

SUBSTITUTE SHEET (RULE 26)

RAD ORIGINAL



- 28 -

```

define RSReg1 '8ff8'
define RSReg2 '8ff9'
define RSReg3 '8ffa'
define RSReg4 '8ffb'
define RSReg5 '8ffc'
define RSReg6 '8ffd'
define RSReg7 '8ffe'
define RSReg8 '8fff'
define RSIN 'ffff'
define RSOUT 'seff8'

;define PORT C initializations
; encoder PORT C Assignments
;
; s = ssi port
; i = input port
; o = output port
;
; 8 - 7 6 5 4 - 3 2 1 0
; s s s s : s o i o
; 1 e 8
;
; 0101 = 5
pc0 = eclkssel (o) ;select clock for Reed Solomon
pc1 = eld (i) ;phase lock detect (0=not locked, 1=locked)
pc2 = rstres (o) ;reset Reed Solomon
pc3 = ebclk (si) ;bit clock
;
; 0000 = 0
pc4 = elrcik (i) ;input pcm samples left/right clock
pc5 = ewclk (si) ;transmit word clock
pc6 = eclk (si) ;input samples word clock
pc7 = esrdata (si) ;input audio pcm sample data
;
; 0000 = 0
pc8 = etdata (so) ;output MUSICAM frame data

define XCODE_PORT_C_M_PCC 'movep #>S01e8,x:<<$FFE1'
define XCODE_PORT_C_M_PCD 'movep #>S0004,x:<<$FFE5'
define XCODE_PORT_C_M_PCDDR 'movep #>S0005,x:<<$FFE3'

; decoder PORT C Assignments
;
; s = ssi port
; i = input port
; o = output port
;
; 8 - 7 6 5 4 - 3 2 1 0
; s s s s : s o o i
; 1 d 8
;
; 0110 = 6
pc0 = dld (i) ;phase lock detect (0=not locked, 1=locked)
pc1 = fclksel (o) ;select clock for Reed Solomon
pc2 = darst (o) ;d-to-a reset line (0 = mute, 1 = audio)
pc3 = dclk (si) ;receive input frame data stream clock
;
; 0000 = 0
pc4 = dwclk (si) ;transmit dac output audio word clock

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 29 -

```

; pc5 = dirclk ; transmit dac audio output left/right clock
; pc6 = dbclk ; decoder bit clock
; pc7 = drdata ; receive input musicam frame data

; 0000 = 0
; pc8 = dsdata (so) ; transmit audio data output to dac

define RDECODE_PORT_C_M_PCC 'movep #>S01d8,x:<<SFFE1'
define RDECODE_PORT_C_M_PCD 'movep #>S0002,x:<<SFFE5'
define RDECODE_PORT_C_M_PCDDR 'movep #>S0006,x:<<SFFE3'

; define PORT B initializations
; encoder PORT B Assignments

; Note: for Digicast port B is a host port
; That means the following definitions are not applicable.

; 14 13 12 - 11 10 9 8 - 7 6 5 4 - 3 2 1 0
; 14 13 12 - 11 10 9 8 - 7 6 5 4 - 3 2 1 0
; 1 1 1 0 1 1 0 0 0 0 0 0 1 1 ** MUSICAM **
; 1 0 1 0 1 1 0 0 0 0 0 0 1 1 ** G722 **
; 1 0 1 0 1 1 0 0 0 0 1 0 1 1 ** G722 **

; 1100 = 0 ** MUSICAM **
; 0100 = 4 ** G722 **

; pb0 = !lb (i) ; loop back
; pb1 = bitrate (i) ; frame bit rate (0=low, 1=high)
; pb2 = coding (o) ; type of data input (0=MUSICAM, 1=G722)
; pb3 = samprate (o) ; PCM sampling rate (0=low, 1=high) ** MUSICAM **
; pb3 = samprate (i) ; HSFTT flag for H221 ** G722 **

; 1111 = f
; pb4 = emus (o) ; encoder MUSICAM led (0=off, 1=lit)
; pb5 = eovrld (c) ; input pcm overload led (0=off, 1=lit: alarm)
; pb6 = e24k (o) ; encoder phase lock loop led (0=off, 1=lit)
; pb7 = wd2 (o) ; watch dog timer

; 1001 = 9
; pb8 = cal (o) ; analog-to-digital converter reset (0=normal, 1=reset)
; pb9 = e0 (i) ; C0 flag for H221 ** G722 **
; pb10 = e1 (i) ; C2 flag for H221 ** G722 **
; pb11 = era15 (o) ; must be set to 1

; 000 = 0 ** MUSICAM **
; 010 = 2 ** G722 **

; pb12 = e3 (i) ; ABIT flag for H221 ** G722 **
; pb13 = e2 (i) ; NOT USED ** MUSICAM **
; pb13 = e2 (o) ; HSTTF flag for H221 ** G722 **
; pb14 = e4 (i) ; NOT USED ** MUSICAM **
; pb14 = e4 (o) ; HSFTT flag for H221 ** G722 **
; pb14 = e4 (i) ; auto status of decoder: 0 go to low sampling/MUSICAM
; ; 1 follow above pins

; Note: for Digicast port B is a host port
; That means the previous definitions are not applicable.

; define port B as a host port
define XCODE_PORT_B_M_PBC 'movep #>S0001,x:<<SFFE0'

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 31 -

; set data so that baral5 (bit 11) is 1

```

define RDECODE_PORT_B_M_PBD      '!!!Digicastmovep #>S0800,x:<<SFFE4'
** MUSICAM **
define RDECODE_PORT_B_M_PBDDR    '!!!Digicastmovep #>Sffbe,x:<<SFFE2'
** G722 **
define FRADPCM_PORT_B_M_PBDDR    '!!!Digicastmovep #>Sdfbe,x:<<SFFE2'

```

; define ssi port initialization for encoder and decoder

```

define XCODE_SSI_M_CRA            'movep #>S6000,x:<<SFFEC'
define XCODE_SSI_M_CRB            'movep #>Sf010,x:<<SFFED'

define RDECODE_SSI_M_CRA          'movep #>S6000,x:<<SFFEC'
define RDECODE_SSI_M_CRB          'movep #>Sf008,x:<<SFFED'

```

; define sci port initialization for encoder and decoder

```

define XCODE_SCI_M_SCR            'movep #>S0002,x:<<SFFFC'
define RDECODE_SCI_M_SCR          'movep #>S0002,x:<<SFFFC'

```

```

; define the setting dsp56002 clock (PLL Control Register)
; 8MHz crystal to run a 40 MHz (5 times 8, so code a 4 below)

```

```

define XCODE_M_PCTL              'movep #>S050004,x:<<SFFFD'
define RDECODE_M_PCTL            'movep #>S050004,x:<<SFFFD'

```

; ENCODER hardware settings for leds and lines

```

; control the encoder devices:
; tested inputs of:

```

```

; host: vector 24
; provides hardware and encoding parameters: none yet
; host: vector 2A
; psycho table parameter id (0 - 31)
; host: vector 2C
; psycho table parameter value for is from host vector 28

```

```

; y:<<SFFFF
; BRAD encode select data type          bit 0 (0=MUSICAM, 1=G722) sw1
; LO/HI encode sampling rate            bit 1 (0=high, 1=low) sw2
; CODAD decode select data type         ; bit 2 (0=MUSICAM, 1=G722) sw3
; MJS/G722 decode sampling rate         ; bit 3 (0=high, 1=low) sw4
; SRAD bit rate                         bit 4 (0=56Kbits, 1=64Kbits) sw5
; 32/48 not used                        ; bit 5 (0=low, 1=high) sw6
; low bit encoder band width code        bit 8 (0=0, 1=1) sw 1 back panel
; high bit encoder band width code        bit 9 (0=0, 1=1) sw 2 back panel
; baud rate code low order bit           bit 10 (0=0, 1=1) sw 3 back panel
; baud rate code middle bit              bit 11 (0=0, 1=1) sw 4 back panel
; baud rate code high order bit          bit 12 (0=0, 1=1) sw 5 back panel
; CRC-16 OLD (0) or NEW (1) ISO          bit 13 (0=old, 1=new) sw 6 back panel

```

```

; !!!Note: for Digicast port B is a host port
; That means the following definitions are not applicable.
; M_PBD (x:<<SFFE4)

```

```

; pb1 = bitrate (i)          bit 1 frame bit rate (0=low, 1=high)
; pb9 = ec (i)              bit 9 C0 flag for H221 ** G722 **

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 32 -

```

pb10 = e1 (i)      bit 10 C2 flag for H221 ** G722 **
pb12 = e3 (i)      bit 12 ABIT flag for H221 ** G722 **
pb13 = e2 (i)      bit 13 NOT USED ** MUSICAM **
pb14 = e4 (i)      bit 14 HSFFT flag for H221 ** G722 **

set outputs of:
!!!Note: for Digicast port B is a host port
That means the following definitions are not applicable.

pb2 = coding (o)    M_PBD (x:<<SFFE4)
                    bit 2 type of data input (0=MUSICAM, 1=G722)
pb3 = samprate (o)  bit 3 PCM sampling rate (0=low, 1=high)
pb4 = emus (o)      bit 4 MUSICAM encoding led (0=off, 1=lit alarm)
pb5 = eovld (o)     bit 5 input pcm overload led (0=off, 1=lit alarm)
pb6 = epl1alm (o)   bit 6 encoding at low sampling led (0=off, 1=lit)
pb7 = wd2 (o)       bit 7 watch dog timer
pb8 = !cal (o)      bit 8 anal-to-digit converter reset (1=normal, 0=reset)
pb11 = era15 (o)    bit 11 must be set to 1
pb13 = e2 (o)       bit 13 HSTTF flag for H221 ** G722 **
pc2 = eg722 (o)     M_PBD (x:<<SFFE5)
                    bit 2 G722 encoding led (0=off, 1=lit alarm)

leds across panel:
!!!Note: for Digicast port B is a host port
That means the following definitions are not applicable.

1. MUSICAM encoding led: x:<<SFFE4 bit 4 (amber)
2. G722 encoding led: x:<<SFFE5 bit 2 (amber)
N/A 9. main phase lock loop led:
10. encoder overload led: x:<<SFFE4 bit 5 (red)
11. encoding low sampling led: x:<<SFFE4 bit 6 (amber)

!CAL: control the encoder analog-to-digital converter reset line

define SET_ADC_RESET ;bclr #0,y:<not_appl
define CLR_ADC_RESET ;bclr #0,y:<not_appl

LD: test the MAIN phase lock loop detect

define LOCK_COUNT '5' ;5 successive locks set the lock led

define TST_SET_PHASE_LOCK_CD ;jset #1,x:<<SFFE5
define TST_CLR_PHASE_LOCK_CD ;jclr #1,x:<<SFFE5
define TST_ON_PHASE_LOCK_LED_XADPCM ;jset #1,x:<<SFFE5
define TST_OFF_PHASE_LOCK_LED_XADPCM ;jclr #1,x:<<SFFE5

band-width:
low order bit of band-width limit code
high order bit of band-width limit code
codes: 00 = level 0 CDQ2000 standard band-widths
       01 = level 1 CDQ2000 standard band-widths
       10 = level 2 CDQ2000 standard band-widths
       11 = level 3 CDQ2000 standard band-widths

define TST_SET_LOW_BAND_WIDTH_CD ;jclr #0,y:<not_appl
define TST_SET_HIGH_BAND_WIDTH_CD ;jclr #0,y:<not_appl
define TST_CLR_LOW_BAND_WIDTH_CD ;jclr #0,y:<not_appl
define TST_CLR_HIGH_BAND_WIDTH_CD ;jclr #0,y:<not_appl

TOGGLE_WATCH_DOG_CD macro

```



- 33 -

```

; encoder host interface watch dog tickle
; see what the host expects for a dog tickle and act accordingly
; if bit M_HF0 (host i/f flag 0) of X:M_HSR (host status register) is set,
;   set bit M_HF2 (host i/f flag 2) of X:M_HCR (host control register)
; else
;   clear bit M_HF2 (host i/f flag 2) of X:M_HCR (host control register)

```

```

    jset    #4,x:<<SF FE9, _watch_dog_00
    bset    #4,x:<<SF FE8
    jmp     <_watch_dog_10

```

```

_watch_dog_00
    bclr    #4,x:<<SF FE8

```

```

_watch_dog_10

```

```

    endm

```

```

INTERRUPT_HOST_CD macro

```

```

; wiggle host interrupt !HACK bit 14 of port b

```

```

    bset    #14,x:<<SF FE4
    nop
    nop
    movep   y:word_out,x:<<SF FEB ; output leds for last frame
    nop
    nop
    bclr    #14,x:<<SF FE4
    endm

```

```

INIT_HOST_VECTORS_CD macro

```

```

; initialize the encoder host vectors with start-up valid settings
; since value is zero, use 30 sub-bands (6750 Hz)

```

```

    move    #>S0,x0
    move    x0,y:host24_word
    move    #>-1,x0
    move    x0,y:host2A_word
    move    #>S0,x0
    move    x0,y:host2C_word
    endm

```

```

GET_SWITCHES_CD macro LOOP

```

```

; copy switches received under host vector interrupt
; bits 0-4 allow user set audio band width by specifying the upper
; sub-band to be considered for bit allocation.
; the range is from 4 (900 Hz) to 30 (6750 Hz)
; Note: 30 is the default if the value is not within the range

```

```

    move    y:host24_word,x0
    move    x0,y:word_in

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 34 -

```

endm

;BITRATE, low/high: get the selected bit rate
define TST_SET_LO_BIT_RATE_CD      'jclr #0,y:<not_appl'
define TST_SET_HI_BIT_RATE_CD      'jclr #0,y:<not_appl'

;CODAD, MUS/G722: get the selected type of decoder input data
define TST_SET_MUSICAM_DATA_CD      'jclr #0,y:<word_in'
;!!!28.8
define TST_SET_G722_DATA_CD         'jset #0,y:<not_appl'
define SET_MUSICAM_DATA_CD          'bclr #0,y:<not_appl'
define SET_G722_DATA_CD             'bclr #0,y:<not_appl'
;!!!28.8

;SDAD, LOW or HIGH: get the selected sampling rate
; choice pairings (A/B) are: 16/24 16/32 16/48 24/32 24/48 32/48
define TST_SET_LO_SAMPLE_RATE_CD    'jclr #0,y:<not_appl'
define TST_SET_HI_SAMPLE_RATE_CD    'jclr #0,y:<not_appl'
define SET_LO_SAMPLE_RATE_CD        'bclr #0,y:<not_appl'
;!!!28.8
define SET_HI_SAMPLE_RATE_CD        'bclr #0,y:<not_appl'
;!!!28.8

;MONSTERC: test whether mono or stereo framing selected
define TST_SET_MONO_STEREO_CD       'jclr #0,y:<not_appl'
define TST_CLR_MONO_STEREO_CD       'jclr #0,y:<not_appl'

;JOINTCE: test for joint stereo framing (if not mono selected above)
define TST_SET_JOINT_STEREO_CD      'jclr #0,y:<not_appl'
define TST_CLR_JOINT_STEREO_CD      'jclr #0,y:<not_appl'

;set which type ISO CRC-16 checksum OLD (0) or NEW (1)
define TST_SET_NEW_ISO_CRC_CD       'jclr #0,y:<not_appl'
define TST_CLR_NEW_ISO_CRC_CD       'jclr #0,y:<not_appl'

;E4: see if decoder is framed or force MUSICAM at LOW sampling rate
define TST_SET_DECODER_FRAMED_CD    'jclr #0,y:<not_appl'
define TST_CLR_DECODER_FRAMED_CD    'jclr #0,y:<not_appl'

;BR0, BR1, BR2: get the ancillary data baud rate
define TST_SET_LOW_BAUD_RATE_CD     'jclr #0,y:<not_appl'
define TST_SET_MID_BAUD_RATE_CD     'jclr #0,y:<not_appl'
define TST_SET_HIGH_BAUD_RATE_CD    'jclr #0,y:<not_appl'
define TST_CLR_LOW_BAUD_RATE_CD     'jclr #0,y:<not_appl'
define TST_CLR_MID_BAUD_RATE_CD     'jclr #0,y:<not_appl'
define TST_CLR_HIGH_BAUD_RATE_CD    'jclr #0,y:<not_appl'

;summary alarm relay: alarm relay associated with alarm LED
define SET_ALARM_RELAY_CD           'bclr #0,y:<not_appl'
define CLR_ALARM_RELAY_CD           'bclr #0,y:<not_appl'
define TST_SET_ALARM_RELAY_CD       'jclr #0,y:<not_appl'

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 35 -

```

define TST_CLR_ALARM_RELAY_CD      'jclr  #0,y:<not_appl'

;define state for all leds on and off for start-up

define OFF_LEDS_CD      'S000000'      ;off if bits set
define ON_LEDS_CD       'S000000'      ;lit if bits clear

;turn leds off:

define OFF_MUSICAM_LED_CD      ;bclr  #0,y:<not_appl
define OFF_G722_LED_CD        ;bclr  #0,y:<not_appl
define OFF_LOW_SAMPLING_LED_CD ;bclr  #0,y:<not_appl
define OFF_OVERLOAD_LED_CD    ;bclr  #1,y:<word_out
define OFF_MONO_LED_CD        ;bclr  #0,y:<not_appl
define OFF_STEREO_LED_CD      ;bclr  #0,y:<not_appl
define OFF_JOINT_LED_CD       ;bclr  #0,y:<not_appl
define OFF_PHASE_LOCK_LED_CD  ;bset  #0,y:<word_out
define OFF_PHASE_LOCK_LED_XADPCM ;bclr  #0,y:<not_appl
define OFF_ALARM_LED_CD       ;bclr  #0,y:<not_appl
define OFF_BITALLOC_LED_CD    ;bclr  #0,y:<not_appl
define OFF_REED_SOL_LED_CD    ;bclr  #2,y:<word_out

;turn leds on:

define ON_MUSICAM_LED_CD      ;bclr  #0,y:<not_appl
define ON_G722_LED_CD        ;bclr  #0,y:<not_appl
define ON_LOW_SAMPLING_LED_CD ;bclr  #0,y:<not_appl
define ON_OVERLOAD_LED_CD    ;bset  #1,y:<word_out
define ON_MONO_LED_CD        ;bclr  #0,y:<not_appl
define ON_STEREO_LED_CD      ;bclr  #0,y:<not_appl
define ON_JOINT_LED_CD       ;bclr  #0,y:<not_appl
define ON_PHASE_LOCK_LED_CD  ;bclr  #0,y:<word_out
define ON_PHASE_LOCK_LED_XADPCM ;bclr  #0,y:<not_appl
define ON_ALARM_LED_CD       ;bclr  #0,y:<not_appl
define ON_BITALLOC_LED_CD    ;bclr  #0,y:<not_appl
define ON_REED_SOL_LED_CD    ;bset  #2,y:<word_out

define SET_LEDS_CD      'movep  y:word_out,y:<<SFFFF'

;.....
;DECODER hardware settings for leds and lines

;control the decoder devices:
; phase lock loop signal line:  M_PBD bit 6

;control the decoder devices:
; tested inputs of:
; y:<<SFFFF
;::: BRAD encode select data type      ;bit 0 (0=MUSICAM, 1=G722) sw1
;::: LO/HI encode sampling rate        ;bit 1 (0=high, 1=low) sw2
;::: CODAD decode select data type     ;bit 2 (0=MUSICAM, 1=G722) sw3
;::: MUS/G722 decode sampling rate     ;bit 3 (0=high, 1=low) sw4
;::: SRAD decode bit rate              ;bit 4 (0=56Kbits, 1=64Kbits) sw5
;::: 32/48 not used                    ;bit 5 (0=low, 1=high) sw6
;::: low bit encoder band width code   ;bit 8 (0=0, 1=1) sw 1 back panel
;::: high bit encoder band width code  ;bit 9 (0=0, 1=1) sw 2 back panel
;::: baud rate code low order bit      ;bit 10 (0=0, 1=1) sw 3 back panel
;::: baud rate code middle bit         ;bit 11 (0=0, 1=1) sw 4 back panel

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 36 -

```

; baud rate code high order bit          bit 12 (0=0, 1=1 sw 5 back panel)
; CRC-16 OLD (0) or NEW (1) ISO          bit 13 (0=cld, 1=new, sw 5 back panel)
; !!!Note: for Digicast port B is a host port
; That means the following definitions are not applicable.
; M_PBD (x:<<SFFE4)
; M_PBD (x:<<SFFE4)
; !LB loop back                          bit 0 (1=not loop back, 0=loop back)
; LD main phase lock loop signal line: bit 6 (1=lock 0=not)
; E2 HSTTF flag for H221 ** G722 **      bit 13
;
; set outputs of:
; !!!Note: for Digicast port B is a host port
; That means the following definitions are not applicable.
; M_PBD (x:<<SFFE4)
; M_PBD (x:<<SFFE4)
; pb1 = bitrate (o)                      bit 1 determined framing bit rate (0=low, 1=high)
; pb2 = coding (o)                       bit 2 type of data to decode (0=MUSICAM, 1=G722)
; pb3 = samprate (o)                     bit 3 determined sampling rate (0=low, 1=high)
; pb4 = 32k (o)                          bit 4 sampling rate low led-9 (0=off, 1=lit)
; pb5 = 48k (o)                          bit 5 sampling rate high led-10 (0=off, 1=lit)
; pb7 = wdt (o)                          bit 7 watch dog timer (0=clear, 1=set)
; pb8 = !darst (o)                       bit 8 digital-to-analog reset (1=normal, 0=reset)
; pb9 = e0 (o)                           bit 9 C0 flag for H221 ** G722 **
; pb10 = e1 (o)                          bit 10 C2 flag for H221 ** G722 **
; pb11 = decra15 (o)                     bit 11 boot top (1) or bottom (0) must be 1
; pb12 = e3 (o)                          bit 12 ABIT flag for H221 ** G722 **
; pb13 = e2 (o)                          bit 13 NOT USED ** MUSICAM **
; pb14 = e4 (o)                          bit 14 HSFFT flag for H221 ** G722 **
; M_PBD (x:<<SFFE5)
; pc2 = alrmrly (o)                      bit 2 alarm relay
;
; leds across panel:
; encode 1. MUSICAM data led:             y:<<SFFFF bit 0 (amber) ***
; encode 2. G722 data led:                y:<<SFFFF bit 1 (amber) ***
; 3. MUSICAM frames led:                  y:<<SFFFF bit 2 (amber)
; 4. G722 input data led:                 y:<<SFFFF bit 3 (amber)
; 5. framing alarm led:                   y:<<SFFFF bit 4 (red)
; 6. main phase lock loop led:            y:<<SFFFF bit 5 (green)
; 7. decoder overload led:                y:<<SFFFF bit 6 (red)
; 8. crc bit error led:                   y:<<SFFFF bit 7 (red)
; encode 9. encoder overload led:          y:<<SFFFF bit 6 (red) ***
; encode 10. main phase lock loop led:     y:<<SFFFF bit 5 (green) ***
; encode 11. low (1) vs hi (0) sampling: y:<<SFFFF bit 0 (amber) ***
; 12. low (1) vs hi (0) sampling:         y:<<SFFFF bit 0 (amber)
;
; !CAL: control the decoder digital-to-analog converter reset line
;
; define SET_DAC_RESET                    'bset #2,x:<<SFFES'
; define CLR_DAC_RESET                    'bclr #2,x:<<SFFES'
;
; !LB: test the loop back
;
; define TST_SET_LOOP_BACK_DCD            'jclr #0,y:<not_app'
; define TST_CLR_LOOP_BACK_DCD            'jclr #0,y:<not_app'
; define TST_SET_LOOP_BACK_FRADPCM        'jclr #0,y:<not_app'
; define TST_CLR_LOOP_BACK_FRADPCM        'jclr #0,y:<not_app'
;
; LD: test the MAIN phase lock loop detect

```



- 37 -

```

define TST_SET_PHASE_LOCK_DCD      ;jset  #0,x:<<SFFE5'
define TST_CLR_PHASE_LOCK_DCD      ;jclr  #0,x:<<SFFE5'

TOGGLE_WATCH_DOG_DCD macro
; encoder host interface watch dog tickle
; see what the host expects for a dog tickle and act accordingly
; if bit M_HF0 (host i/f flag 0) of X:M_HSR (host status register) is set,
;   set bit M_HF2 (host i/f flag 2) of X:M_HCR (host control register)
; else
;   clear bit M_HF2 (host i/f flag 2) of X:M_HCR (host control register).

    jset    #4,x:<<SFFE9,_watch_dog_00
    bset    #4,x:<<SFFE8
    jmp     <_watch_dog_10

_watch_dog_00
    bclr    #4,x:<<SFFE8

_watch_dog_10
    endm

INTERRUPT_HOST_DCD macro
; wiggle host interrupt !HACK bit 14 of port b

    bset    #14,x:<<SFFE4
    nop
    nop
    movep    y:word_out,x:<<SFFEB    ;output leds for last frame
    nop
    nop
    bclr    #14,x:<<SFFE4

    endm

INIT_HOST_VECTORS_DCD macro
; initialize the encoder host vectors with start-up valid settings

    move     #>$0,x0
    move     x0,y:host24_word

    endm

GET_SWITCHES_DCD macro LOOP
; copy switches received under host vector interrupt

    move     y:host24_word,x0
    move     x0,y:word_in

    endm

;BRAD,low/high: get the selected bit rate

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 38 -

```

define TST_SET_AUTO_BIT_RATE_DCD      jclr  #0.y:<not_appl
define TST_CLR_AUTO_BIT_RATE_DCD      jclr  #0.y:<not_appl
define TST_SET_AUTO_BIT_RATE_FRADPCM  jclr  #0.y:<not_appl
define TST_CLR_AUTO_BIT_RATE_FRADPCM  jclr  #0.y:<not_appl
define TST_SSET_AUTO_BIT_RATE_FRADPCM jclr  #0.y:<not_appl
define TST_SCLR_AUTO_BIT_RATE_FRADPCM jclr  #0.y:<not_appl
define TST_SET_LO_BIT_RATE_DCD        jclr  #0.y:<not_appl
define TST_SET_HI_BIT_RATE_DCD        jclr  #0.y:<not_appl
define TST_SET_LO_BIT_RATE_FRADPCM    jclr  #0.y:<not_appl
define TST_SET_HI_BIT_RATE_FRADPCM    jclr  #0.y:<not_appl
;!!!28.8
define SET_LO_BIT_RATE_DCD             bclr  #0.y:<not_appl
define SET_HI_BIT_RATE_DCD            bclr  #0.y:<not_appl
;!!!28.8
;CODAD,MUS/G722: get the selected type of decoder input data
define TST_SET_AUTO_CODED_DATA_DCD    jclr  #0.y:<not_appl
define TST_CLR_AUTO_CODED_DATA_DCD    jclr  #0.y:<not_appl
define TST_SET_AUTO_CODED_DATA_FRADPCM jclr  #0.y:<not_appl
define TST_CLR_AUTO_CODED_DATA_FRADPCM jclr  #0.y:<not_appl
define TST_SSET_AUTO_CODED_DATA_FRADPCM jclr  #0.y:<not_appl
define TST_SCLR_AUTO_CODED_DATA_FRADPCM jclr  #0.y:<not_appl
define TST_SET_MUSICAM_DATA_DCD        jclr  #0.y:<not_appl
define TST_SET_G722_DATA_DCD          jclr  #0.y:<not_appl
define TST_SET_MUSICAM_DATA_FRADPCM    jclr  #0.y:<not_appl
define TST_SET_G722_DATA_FRADPCM      jclr  #0.y:<not_appl
;!!!28.8
define SET_MUSICAM_DATA_DCD           bclr  #0.y:<not_appl
define SET_G722_DATA_DCD              bclr  #0.y:<not_appl
;!!!28.8
;STAD,low or high: get the selected sampling rate
; choice pairings (A/B) are: 16/24 16/32 16/48 24/32 24/48 32/48
define TST_SET_AUTO_SAMPLE_RATE_DCD   jclr  #0.y:<not_appl
define TST_CLR_AUTO_SAMPLE_RATE_DCD   jclr  #0.y:<not_appl
define TST_SET_LO_SAMPLE_RATE_DCD     jclr  #0.y:<not_appl
define TST_SET_HI_SAMPLE_RATE_DCD     jclr  #0.y:<not_appl
;!!!28.8
define SET_LO_SAMPLE_RATE_DCD         bclr  #0.y:<not_appl
define SET_HI_SAMPLE_RATE_DCD         bclr  #0.y:<not_appl
;!!!28.8
;E4: inform the encoder:
define SET_DECODER_FRAMED_DCD         bclr  #0.y:<not_appl
;DSW7: mute the decoder output
define TST_SET_MUTE_OUTPUT_DCD        jclr  #0.y:<not_appl
define TST_CLR_MUTE_OUTPUT_DCD        jclr  #0.y:<not_appl
;DSW8,DSW9: test the mono output channel requirements
define TST_SET_MONO_ONE_CHANNEL_DCD   jclr  #0.y:<not_appl
define TST_CLR_MONO_ONE_CHANNEL_DCD   jclr  #0.y:<not_appl
define TST_SET_MONO_LEFT_OR_RIGHT_DCD jclr  #0.y:<not_appl
define TST_CLR_MONO_LEFT_OR_RIGHT_DCD jclr  #0.y:<not_appl

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 39 -

;to be activated sometime in CDQ1000

```

define TST_SET_FADE_OUTPUT_DCD      'jclr  #0,y:<not_appl'
define TST_CLR_FADE_OUTPUT_DCD      'jclr  #0,y:<not_appl'
define TST_SET_FADE_UP_DCD          'jclr  #0,y:<not_appl'
define TST_SET_FADE_DOWN_DCD        'jclr  #0,y:<not_appl'
define FADE_INCREMENT '1'           ;2 Db per frame
define FADE_SOFTTEST '40'           ;max of down: 80 Db
define FADE_START_UP '20'           ;max of start up 40 Db
define FADE_FRAMES '2'              ;fade every N frames

```

;LINSEL0,LINSEL1: test if line 1 and/or line 2 is selected

```

define TST_SET_LINE_1_SELECT_DCD    'jclr  #0,y:<not_appl'
define TST_SET_LINE_2_SELECT_DCD    'jclr  #0,y:<not_appl'
define TST_CLR_LINE_1_SELECT_DCD    'jset  #0,y:<not_appl'
define TST_CLR_LINE_2_SELECT_DCD    'jset  #0,y:<not_appl'

```

;DIAGNOST (ANCELDTA): test whether diagnostics programming is to be executed

```

define TST_SET_DIAGNOSTICS_DCD      'jclr  #0,y:<not_appl'
define TST_CLR_DIAGNOSTICS_DCD      'jclr  #0,y:<not_appl'

```

;BR0,BR1,BR2: get the ancillary data baud rate

```

define TST_SET_LOW_BAUD_RATE_DCD    'jclr  #0,y:<not_appl'
define TST_SET_MID_BAUD_RATE_DCD    'jclr  #0,y:<not_appl'
define TST_SET_HIGH_BAUD_RATE_DCD   'jclr  #0,y:<not_appl'
define TST_CLR_LOW_BAUD_RATE_DCD    'jclr  #0,y:<not_appl'
define TST_CLR_MID_BAUD_RATE_DCD    'jclr  #0,y:<not_appl'
define TST_CLR_HIGH_BAUD_RATE_DCD   'jclr  #0,y:<not_appl'

```

;BR0,BR1,BR2: get diagnostics code when DIAGNOST (currently ANCELDTA) is set
; dip switch interpretations for diagnostic operation

```

define TST_SET_LOW_DIAG_CODE_DCD    'jclr  #0,y:<not_appl'
define TST_SET_MID_DIAG_CODE_DCD    'jclr  #0,y:<not_appl'
define TST_SET_HIGH_DIAG_CODE_DCD   'jclr  #0,y:<not_appl'
define TST_CLR_LOW_DIAG_CODE_DCD    'jclr  #0,y:<not_appl'
define TST_CLR_MID_DIAG_CODE_DCD    'jclr  #0,y:<not_appl'
define TST_CLR_HIGH_DIAG_CODE_DCD   'jclr  #0,y:<not_appl'

```

;summary alarm relay: alarm relay associated with alarm LED

```

define SET_ALARM_RELAY_DCD           'bclr  #0,y:<not_appl'
define CLR_ALARM_RELAY_DCD           'bclr  #0,y:<not_appl'
define TST_SET_ALARM_RELAY_DCD       'jclr  #0,y:<not_appl'
define TST_CLR_ALARM_RELAY_DCD       'jclr  #0,y:<not_appl'

```

;define state for all leds on and off for start-up

```

define OFF_LEDS_DCD '500' ;off if bits set'
define ON_LEDS_DCD  '5ff' ;li: if bits clear'

```

;turn leds off:

```

define OFF_FRAME_LED_DCD      'bclr  #1,y:<word_out'
define OFF_CRC_ERROR_LED_DCD 'bclr  #2,y:<word_out'
define OFF_OVERLOAD_LED_DCD  'bclr  #3,y:<word_out'
define OFF_PHASE_LOCK_LED_DCD 'bset  #4,y:<word_out'

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 40 -

```

define OFF_REED_SOL_LED_DCD          'bclr #5,y:<word_out'
define OFF_LO_BIT_RATE_LED_DCD      ';bclr #0,y:<not_appl'
define OFF_HI_BIT_RATE_LED_DCD      ';bclr #0,y:<not_appl'
define OFF_MUSICAM_LED_DCD          ';bclr #0,y:<not_appl'
define OFF_G722_LED_DCD             ';bclr #0,y:<not_appl'
define OFF_PHASE_LOCK_LED_FRADPCM    ';bclr #0,y:<not_appl'
OFF_PHASE_LOCK_LED_MACRO_FRADPCM macro
bclr #5,x:<Eram_Mem          ;turn off red led
move x:<Eram_Mem,x0
movep x0,y:<<$FFFF
endm
OFF_OVERLOAD_LED_MACRO_FRADPCM macro
bclr #6,x:<Eram_Mem          ;turn off overload led
movep x:Eram_Mem,y:<<$FFFF
endm
define OFF_LO_SAMPLE_RATE_LED_DCD    ';bclr #0,y:<not_appl'
define OFF_HI_SAMPLE_RATE_LED_DCD    ';bclr #0,y:<not_appl'
define OFF_MONO_LED_DCD              ';bclr #0,y:<not_appl'
define OFF_STEREO_LED_DCD            ';bclr #0,y:<not_appl'
define OFF_JOINT_LED_DCD             ';bclr #0,y:<not_appl'
define OFF_ALARM_LED_DCD             ';bclr #0,y:<not_appl'

;turn leds on:

define ON_FRAME_LED_DCD              'bset #1,y:<word_out'
define ON_CRC_ERROR_LED_DCD          'bset #2,y:<word_out'
define ON_OVERLOAD_LED_DCD           'bset #3,y:<word_out'
define ON_PHASE_LOCK_LED_DCD         'bclr #4,y:<word_out'
define ON_REED_SOL_LED_DCD           'bset #5,y:<word_out'

define ON_LO_BIT_RATE_LED_DCD        ';bclr #0,y:<not_appl'
define ON_HI_BIT_RATE_LED_DCD        ';bclr #0,y:<not_appl'
define ON_MUSICAM_LED_DCD            ';bclr #0,y:<not_appl'
define ON_G722_LED_DCD              ';bclr #0,y:<not_appl'
define ON_PHASE_LOCK_LED_FRADPCM     ';bclr #0,y:<not_appl'
ON_PHASE_LOCK_LED_MACRO_FRADPCM macro
bset #5,x:<Eram_Mem          ;turn on red led
move x:<Eram_Mem,x0
movep x0,y:<<$FFFF
endm
ON_OVERLOAD_LED_MACRO_FRADPCM macro
bset #6,x:<Eram_Mem          ;turn on overload led
movep x:Eram_Mem,y:<<$FFFF
endm
define ON_LO_SAMPLE_RATE_LED_DCD     ';bclr #0,y:<not_appl'
define ON_HI_SAMPLE_RATE_LED_DCD     ';bclr #0,y:<not_appl'
define ON_MONO_LED_DCD               ';bclr #0,y:<not_appl'
define ON_STEREO_LED_DCD             ';bclr #0,y:<not_appl'
define ON_JOINT_LED_DCD              ';bclr #0,y:<not_appl'
define ON_ALARM_LED_DCD              ';bclr #0,y:<not_appl'

define SET_LEDS_DCD                  'movep y:word_out,y:<<$FFFF'

define TST_SET_CRC_ERROR_DCD         'jclr #0,y:<not_appl'
define TST_CLR_CRC_ERROR_DCD         'jclr #0,y:<not_appl'

;define macros for getting the encoder and decoder external switches
GET_BIT_RATE_CD macro

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 41 -

```

; encoder interpret the external switches for the framing bit rate

        move    #>RATE_LO,x0          ;start with lower KBit rate
;!!!28.8: force low bit rate
;!!!!    TST_SET_LO_BIT_RATE_CD,_grte_a
;!!!!    move    #>RATE_HI,x0          ;otherwise, use higher KBit rate
;!!!!
;!!!!_grte_a
        move    x0,x:tstrate          ;set selected rate

        endm

GET_FRAME_TYPE_CD macro
; micro encoder only handles mono frame type

        move    #>MONO,x0
        move    x0,x:tstfrme

;;; determine the NEW or OLD ISO CRC-16 specification
;!!
;!!    bclr    #CRC_OLD_vs_NEW,y:<stereo ;0=OLD ISO specification
;!!                                           ;1=NEW ISO specification
;!!    TST_CLR_NEW_ISO_CRC_CD,_gtyp_a    ;if not use NEW CRC, done
;!!
; MiniCodec board FORCE new ISO crc

        bset    #CRC_OLD_vs_NEW,y:<stereo ;1=NEW ISO specification

;!!!_gtyp_a

; default to old CCS CDQ1000's

        bset    #0,x:tstoccs          ;1=old CCS CDQ2000's

        endm

GET_CODE_TYPE_CD macro

; encoder interpret the external switches for the type of coded output
; MUSICAM frames or G722

;!!!28.8: force MUSICAM
;!!!!    TST_SET_MUSICAM_DATA_CD,_gcde_a
;!!!!    bset    #0,x:tstcode          ;indicate G722 output
;!!!!    OFF_MUSICAM_LED_CD            ;turn off MUSICAM indicator
;!!!!    OFF_LOW_SAMPLING_LED_CD      ;turn off low sampling rate indicator
;!!!!    ON_G722_LED_CD               ;turn on G722 indicator
;!!!!    SET_G722_DATA_CD             ;set line for encoder G722
;!!!!    jmp     <_gcde_b

;!!!_gcde_a
        ON_MUSICAM_LED_CD              ;turn on MUSICAM indicator
        OFF_G722_LED_CD                ;turn off G722 indicator
        SET_MUSICAM_DATA_CD            ;set line for encoder MUSICAM

;!!!_gcde_b
        endm

```



- 42 -

GET_SAMPLE_RATE_CD macro

; micro encoder handles low and high sampling rates

```

;!!! 28.8: force low sample rate
;!!! TST_SET_LO_SAMPLE_RATE_CD, _gsmp_a
;!!! bset #0,x:tstsmpl ;indicate high K sampling rate
;!!! OFF_LO_SAMPLING_LED_CD ;turn off low sampling rate indicator
;!!! SET_HI_SAMPLE_RATE_CD ;set line for high sampling rate
;!!! jmp <_gsmp_b
;!!!
;!!! _gsmp_a
;!!! TST_SET_G722_DATA_CD, _gsmp_b ;do not turn on if G722
;!!! ON_LO_SAMPLING_LED_CD ;turn on low sampling rate indicator
;!!! SET_LO_SAMPLE_RATE_CD ;set line for low sampling rate
;!!!
;!!! _gsmp_b
;!!! endm

```

GET_BAND_WIDTH_CD macro

; encoder interpret the external switches for the band-width code
; to set band-width based on frame bit rate and type of framing

```

;!!! TST_CLR_LOW_BAND_WIDTH_CD, _gbnd_a ;check switch to interpret as 0
;!!! bset #0,x:tstband ;set the band width code low bit on
;!!!
;!!! _gbnd_a
;!!! TST_CLR_HIGH_BAND_WIDTH_CD, _gbnd_b ;check switch to interpret as 0
;!!! bset #1,x:tstband ;set the band width code high bit on
;!!!
;!!! _gbnd_b
;!!!
; bits 0-4 allow user set audio band width by specifying the upper
; sub-band to be considered for bit allocation.
; the range is from 4 (900 Hz) to 30 (6750 Hz)
; Note: 30 is the default if the value is not within the range
;!!!
;!!! move y:word_in,x0 ;get sub-bands for y:<usedsb
;!!! move x0,x:tstband ;put value in the new i/p
;!!! move x0,y:bandwidth ;& put value in the current
;!!!
;!!! endm

```

GET_BAUD_RATE_CD macro

; encoder interpret the external switches to get ancillary data baud rate

```

;!!! TST_CLR_LOW_BAUD_RATE_CD, _gbaud_a ;check switch to interpret as 0
;!!! bset #0,x:tstbaud ;set the baud rate low bit on
;!!!
;!!! _gbaud_a
;!!! TST_CLR_MID_BAUD_RATE_CD, _gbaud_b ;check switch to interpret as 0
;!!! bset #1,x:tstbaud ;set the baud rate middle bit on
;!!!
;!!! _gbaud_b
;!!! TST_CLR_HIGH_BAUD_RATE_CD, _gbaud_c ;check switch to interpret as 0
;!!! bset #2,x:tstbaud ;set the baud rate high bit on
;!!!
;!!! _gbaud_c
;!!!

```

BAD ORIGINAL

- 43 -

```

    endr

; decoder external switch macros
GET_BIT_RATE_DCD macro

; decoder interpret the external switches for the framing bit rate
; begin with raw code for lower framing bit rate, clear auto select flag

    move    #>RATE_LO,x0
; 28.8: force low bit rate
    bclr    #AUTO_SELECT_BIT_RATE,y:<ctlflgs
    move    #autoRate,r0          ;addr of curr bit auto select state
; if not auto select switch is set, go by the selected switch setting
    TST_CLR_AUTO_BIT_RATE_DCD,_grte_c ;if not auto select, test other sw
; if in loop back, set the bit rate to high Kbits
    TST_CLR_LOOP_BACK_DCD,_grte_a    ;if not loop, continue
    move    #>RATE_HI,x0            ;set higher Kbits raw code
    jmp     <_grte_e                ;install chosen bit rate
;_grte_a
; see if already in auto select bit rate
    jset     #0,x:(r0),_grte_b        ;if already in auto, skip next 2 stmts
; set save code as in auto select bit rate and indicate switch changes
    bset     #0,x:(r0)                ;bit 0 = 1 = AUTO_SELECT
    bset     #4,y:<not_appl           ;indicate a switch change
;_grte_b
; set control flag to perform auto select of bit rate
    bset     #AUTO_SELECT_BIT_RATE,y:<ctlflgs
    bset     #0,x:autoSel
    move     y:frmrate,x0            ;use last rate to start
    jmp     <_grte_e
; set the bit rate as selected by the switch
;_grte_c
; see if currently in auto select bit rate
    jclr     #0,x:(r0),_grte_d        ;if not in auto, skip next 2 stmts
; clear save code as NOT in auto select bit rate and indicate switch changes
    bclr     #0,x:(r0)                ;bit 0 = 0 = NOT AUTO_SELECT
    bset     #4,y:<not_appl           ;indicate a switch change
;_grte_d

```



- 44 -

```

; see if low or high bit rate selected, if 0, keep lower Kbit rate
TST SET_LO_BIT_RATE_DCD, _grte_e
move    #>RATE_HI, x0           ; otherwise, use higher Kbit rate
; _grte_e
move    x0, x: tstrate          ; set selected rate
endm

GET_FRAME_TYPE_DCD macro
; decoder interpret the external switches for the frame type
; (not applicable)
; however, set the current mono frame output channel parameter
; clear the mono out both channels flag and set the flag if needed
bset    #MONO_OUT_BOTH, y: <ctlflgs    ; mono out both channels
TST CLR_MONO_ONE_CHANNEL_DCD, _gfrm_a
bclr    #MONO_OUT_BOTH, y: <ctlflgs    ; mono out one channel
_gfrm_a
; clear the mono output one channel flag indicating LEFT
; and set the flag to the RIGHT channel if needed
bclr    #MONO_OUT_CHANNEL, y: <ctlflgs ; mono one channel out LEFT
TST CLR_MONO_LEFT_OR_RIGHT_DCD, _gfrm_b
bset    #MONO_OUT_CHANNEL, y: <ctlflgs ; mono one channel out RIGHT
_gfrm_b
endm

GET_CODE_TYPE_DCD macro
; decoder interpret the external switches for the type of coded input
; MUSICAM frames or G722
; starts out as MUSICAM (default), clear auto select flag
; 128.8: force MUSICAM
bclr    #AUTO_SELECT_DATA_TYPE, y: <ctlflgs
move    #autocode, r0
; if not auto select switch is set, go by the selected switch setting
TST CLR_AUTO_CODED_DATA_DCD, _gcde_b
; if in loop back, leave the data type as MUSICAM
TST SET_LOOP_BACK_DCD, _gcde_d ; if in loop, done selection
; see if already in auto select code type
jset    #0, x: (r0, _gcde_a      ; if already in auto, skip next 2 stmts
; set save code as in auto select code type and indicate switch changes

```



- 45 -

```

bset    #0,x:(r0)          ;bit 0 = 1 = AUTO SELECT
bset    #4,y:<not_appl      ;indicate a switch change

_gcde_a
; set control flag to perform auto select of bit rate
bset    #AUTO_SELECT_DATA_TYPE,y:<ctlflgs
bset    #0,x:autosei
; set to auto select, continue with previous type of coded data
move    y:inputcde,x0
move    x0,x:tstcde        ;indicate last input type
jmp     <_gcde_d

_gcde_b
; see if currently in auto select code type
jclr    #0,x:(r0),_gcde_c   ;if not in auto, skip next 2 stmts
; clear save code as NOT in auto select code type and indicate switch changes
bclr    #0,x:(r0)          ;bit 0 = 0 = NOT AUTO SELECT
bset    #4,y:<not_appl      ;indicate a switch change

_gcde_c
TST_SET_MUSICAM_DATA_DCD,_gcde_d
bset    #0,x:tstcde        ;indicate G722 input

_gcde_d
; indicate the switch selection to encoder for data type
TST_SET_ENCODE_G722_DATA_DCD,_gcde_e   ;if G722, set that for encoder
SET_ENCODE_MUSICAM_DATA_DCD            ;tell encoder MUSICAM
jmp     <_gcde_f

_gcde_e
SET_ENCODE_G722_DATA_DCD                ;tell encoder G722

_gcde_f
endm

GET_SAMPLE_RATE_DCD macro
; decoder interpret the external switches for the sampling rate
; if select switch is set, see which type of coded data is being input
; begin with the code for low sampling KHz rate, clear auto select flag
move    #0,x0
; 28.8: force low sample rate
bclr    #AUTO_SELECT_SAMPLE_RATE,y:<ctlflgs
move    #autosmpl,r0
; if not auto select switch is set, go by the selected switch setting

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 46 -

```

TST CLR_AUTO_SAMPLE_RATE_DCD, _gsmp_b ;if not auto select, test other sw
;if in loop back, leave the low sampling rate selected
TST SET_LOOP_BACK_DCD, _gsmp_d ;if in loop, done selection
;see if already in auto select sampling rate
jset #0,x:(r0), _gsmp_a ;if already in auto, skip next 2 stmts
;set save code as in auto select sampling rate and indicate switch changes
bset #0,x:(r0) ;bit 0 = 1 = AUTO SELECT
bset #4,y:<not_appl ;indicate a switch change
_gsmp_a
;set control flag to perform auto select of sampling rate
bset #AUTO_SELECT_SAMPLE_RATE,y:<ctrlflgs
bset #0,x:autosel
move y:smpirte,x0 ;use last sampling rate to start
jmp <_gsmp_d
; set the sampling rate as selected by the switch
_gsmp_b
;see if currently in auto select sampling rate
jclr #0,x:(r0), _gsmp_c ;if not in auto, skip next 2 stmts
;clear save code as NOT in auto select sampling rate and indicate switch cha
bclr #0,x:(r0) ;bit 0 = 0 = NOT AUTO SELECT
bset #4,y:<not_appl ;indicate a switch change
_gsmp_c
TST SET_LO_SAMPLE_RATE_DCD, _gsmp_d
move #>1,x0 ;otherwise, use high rate
_gsmp_d
move x0,x:tstsmpl
;indicate the switch selection to encoder for data sampling rate
TST SET_ENCODE_HI_SAMP_RATE_DCD, _gsmp_e ;if high rate, set for encoder
SET_ENCODE_LO_SAMPLE_RATE_DCD ;tell encoder low sampling rate
jmp <_gsmp_f
_gsmp_e
SET_ENCODE_HI_SAMPLE_RATE_DCD ;tell encoder high sampling rate
_gsmp_f
endm

GET_BAUD_RATE_DCD macro
; decoder interprets the external switches to get ancillary data baud rate

```

- 47 -

```

: : : TST_CLR_LOW_BAUD_RATE_DCD, _gbaud_a ;check switch to interpret as 0
: : : bset #0,x:tstbaud ;set the baud rate low bit on
: : : _gbaud_a
: : : TST_CLR_MID_BAUD_RATE_DCD, _gbaud_b ;check switch to interpret as 0
: : : bset #1,x:tstbaud ;set the baud rate middle bit on
: : : _gbaud_b
: : : TST_CLR_HIGH_BAUD_RATE_DCD, _gbaud_c ;check switch to interpret as 0
: : : bset #2,x:tstbaud ;set the baud rate high bit on
: : : _gbaud_c
: : : endm

GET_METHOD_OPERATION_DCD macro
: : : decoder get external switches for method of operation: NORMAL vs DIAGNOSTIC
: : : endm

GET_DIAGNOSTICS_DCD macro
: : : decoder get external switches for diagnostic operation: NORMAL vs DIAGNOSTIC
: : : ; if switch set for normal operation, skip rest of this interpretation
: : : TST_CLR_DIAGNOSTICS_DCD, _gdiag_c ;switch set for normal or diagnostics
: : : ; set the diagnostic code bits
: : : TST_CLR_LOW_DIAG_CODE_DCD, _gdiag_a ;check switch to interpret as 0
: : : bset #0,x:tstmeth ;set diagnostic code low bit on
: : : _gdiag_a
: : : TST_CLR_MID_DIAG_CODE_DCD, _gdiag_b ;check switch to interpret as 0
: : : bset #1,x:tstmeth ;set diagnostic code middle bit on
: : : _gdiag_b
: : : TST_CLR_HIGH_DIAG_CODE_DCD, _gdiag_c ;check switch to interpret as 0
: : : bset #2,x:tstmeth ;set diagnostic code high bit on
: : : _gdiag_c
: : : endm

VERIFY_AUTO_SAMPLE macro
: : : ;:Digicast: NOT APPLICABLE
: : : endm

: : : ;for CDQ2012 start with flag set to decode MPEG-ISO frames:
: : : bit 0: 0 = MPEG-ISC
: : : : 1 = old CCS CDQ's
: : : bit 0: 0 = MPEG-ISO at 24000 sampling
: : : : 1 = old CDQ1000 (MICRO) frames at 24000 sampling

TOO_MANY_SYNC_ERRORS_DCD macro

```



- 48 -

how to handle the set of the REFRAME flag after too many successive
sync pattern failures
always do old CCS CDQ's

```

bset    #0,y:oldccs          ;only handle old CCS CDQ's
bset    #1,y:oldccs          ;old CCS CDQ frms @ 14.4 K sampl
jmp      <restart             ;restart, as old CCS CDQ's

endm

```

TOO_MANY_BIT_ERRORS_DCD macro

how to handle the set of the REFRAME flag after too many successive
CRC-16 bit errors
if the oldccs bit is not set, switch from MPEG-ISO to old CCS CDQ's
if old CCS has already been tried, restore MPEG-ISO and reframe

```

move    #oldccs,r5           ;to test oldccs flag (bit 0)
nop                                           ; 0 = MPEG-ISO
                                           ; 1 = old CCS

_old_ccs

;try decoding frames from older CCS CDQ's units

bset    #0,y:oldccs          ;set old CCS flag
;!!!dbg
nop
nop
nop
nop
nop
;!!!dbg
jmp      <reframe             ;reframe, try old CCS

endm

```

This code handles the special ancillary data problem when frames have
too many encoded according to the decoder baud rate and the frames also
have the old ISO (CCS) CRC-16 checksum algorithm for protection.
This condition occurs when trying to determine if the stream of frames is
from an old CCS CDQ2000 and are two channel frames at low bit rates or is
the stream from a new CCS CDQ with MPEG-ISO frames but are protected
using the old ISO (CCS) CRC-16 algorithm.

TOO_MANY_DATA_ERRORS_DCD macro

old CDQ1000 mono frames @ 24000 sampling do not apply to this problem

```

jset    #1,y:(r1),_tdata_10    ;if old CDQ1000, skip over to continue

;if too many errors, reframe using the opposite old CCS vs MPEG-ISO with
;low bit rate two channel frames

jset    #0,y:(r1),_tdata_00    ;if doing old CCS, go switch to ISO
bset    #0,y:oldccs            ;switch to try old CCS decoding
jmp      <reframe               ;reframe

_tdata_00
bset    #0,y:oldccs            ;switch to try MPEG-ISO decoding
jmp      <restart               ;restart

```



- 49 -

```

_tdata_10
    endm

;define ancillary data baud rates and max byte counts per frame:
;!!!28.8
;    14400 sampling rate @ 80 msecs
;!!!28.8
;    16000 sampling rate @ 72 msecs
;    24000 sampling rate @ 48 msecs
;    32000 sampling rate @ 36 msecs
;    48000 sampling rate @ 24 msecs
;    (baud rate * milliseconds = bits received
;    bits received then promoted to next even 8-bits to yeild max bytes)

;M_SCCRnnn (see pages 11-22 & 11-31) =
;    ((32,000,000 / (64 * nnn)) - 1) (result rounded & converted to hex)
;    where 32,000,000 is crystal, nnn = baud rate

;define BAUD300      'C'      ;dip switch code for 300 baud
;define M_SCCR300    'S662'   ;set clock for 300 baud rate
;!!!28.8
;define BYTES300_16  '3'      ;3 bytes (24.0 bits ==> 24 bits)
;define BYTES300_24  '3'      ;3 bytes (24.0 bits ==> 24 bits)
;define BYTES300_16  '3'      ;3 bytes (21.6 bits ==> 24 bits)
;define BYTES300_24  '2'      ;2 bytes (14.4 bits ==> 16 bits)
;!!!28.8
;define BYTES300_32  '2'      ;2 bytes (10.8 bits ==> 16 bits)
;define BYTES300_48  '1'      ;1 byte (7.2 bits ==> 8 bits)

;define BAUD1200     '1'      ;dip switch code for 1200 baud
;define M_SCCR1200   'S1a0'   ;set clock for 1200 baud rate
;!!!28.8
;define BYTES1200_16 '12'     ;11 bytes (96.0 bits ==> 96 bits)
;define BYTES1200_24 '12'     ;12 bytes (96.0 bits ==> 96 bits)
;define BYTES1200_16 '11'     ;11 bytes (86.4 bits ==> 88 bits)
;define BYTES1200_24 '8'      ;8 bytes (57.6 bits ==> 64 bits)
;!!!28.8
;define BYTES1200_32 '6'      ;6 bytes (43.2 bits ==> 48 bits)
;define BYTES1200_48 '4'      ;4 bytes (28.8 bits ==> 32 bits)

;define BAUD2400     '2'      ;dip switch code for 2400 baud
;define M_SCCR2400   'Scf'    ;set clock for 2400 baud rate
;!!!28.8
;define BYTES2400_16 '24'     ;24 bytes (192.0 bits ==> 192 bits)
;define BYTES2400_24 '24'     ;24 bytes (192.0 bits ==> 192 bits)
;define BYTES2400_16 '22'     ;22 bytes (172.8 bits ==> 176 bits)
;define BYTES2400_24 '15'     ;15 bytes (115.2 bits ==> 120 bits)
;!!!28.8
;define BYTES2400_32 '11'     ;11 bytes (86.4 bits ==> 88 bits)
;define BYTES2400_48 '8'      ;8 bytes (57.6 bits ==> 64 bits)

;define BAUD3600     '3'      ;dip switch code for 3600 baud
;define M_SCCR3600   'S8a'    ;set clock for 3600 baud rate
;!!!28.8
;define BYTES3600_16 '36'     ;36 bytes (288.0 bits ==> 288 bits)
;define BYTES3600_24 '36'     ;36 bytes (288.0 bits ==> 288 bits)
;define BYTES3600_16 '33'     ;33 bytes (259.2 bits ==> 264 bits)

```

SUBSTITUTE SHEET (RULE 26)

RAD ORIGINAL



- 50 -

```

define BYTES3600_24 '22' ;22 bytes (172.8 bits ==> 176 bits)
;:::28.8
define BYTES3600_32 '17' ;17 bytes (129.6 bits ==> 136 bits)
define BYTES3600_48 '11' ;11 bytes (86.4 bits ==> 88 bits)

define BAUD4800 '4' ;dip switch code for 4800 baud
define M_SCCR4800 'S68' ;set clock for 4800 baud rate
;:::28.8
define BYTES4800_16 '48' ;48 bytes (384.0 bits ==> 384 bits)
define BYTES4800_24 '48' ;48 bytes (384.0 bits ==> 384 bits)
define BYTES4800_16 '44' ;44 bytes (345.6 bits ==> 352 bits)
;:::28.8
define BYTES4800_24 '29' ;29 bytes (230.4 bits ==> 232 bits)
;:::28.8
define BYTES4800_32 '22' ;22 bytes (172.8 bits ==> 176 bits)
define BYTES4800_48 '15' ;15 bytes (115.2 bits ==> 120 bits)

define BAUD38400 '5' ;dip switch code for 38400 baud
define M_SCCR38400 'Sc' ;set clock for 38400 baud rate
;:::28.8
define BYTES38400_16 '384' ;384 bytes (3072.0 bits ==> 3072 bits)
define BYTES38400_24 '384' ;384 bytes (3072.0 bits ==> 3072 bits)
define BYTES38400_16 '346' ;346 bytes (2764.8 bits ==> 2768 bits)
;:::28.8
define BYTES38400_24 '231' ;231 bytes (1843.2 bits ==> 1848 bits)
;:::28.8
define BYTES38400_32 '173' ;173 bytes (1382.4 bits ==> 1384 bits)
define BYTES38400_48 '116' ;116 bytes (921.6 bits ==> 928 bits)

define BAUD9600 '6' ;dip switch code for 9600 baud
define M_SCCR9600 'S33' ;set clock for 9600 baud rate
;:::28.8
define BYTES9600_16 '96' ;96 bytes (768.0 bits ==> 768 bits)
define BYTES9600_24 '96' ;96 bytes (768.0 bits ==> 768 bits)
define BYTES9600_16 '87' ;87 bytes (691.2 bits ==> 696 bits)
;:::28.8
define BYTES9600_24 '58' ;58 bytes (460.8 bits ==> 464 bits)
;:::28.8
define BYTES9600_32 '44' ;44 bytes (345.6 bits ==> 352 bits)
define BYTES9600_48 '29' ;29 bytes (230.4 bits ==> 232 bits)

define BAUD19200 '7' ;dip switch code for 19200 baud
define M_SCCR19200 'S19' ;set clock for 19200 baud rate
;:::28.8
define BYTES19200_16 '192' ;192 bytes (1536.0 bits ==> 1536 bits)
define BYTES19200_24 '192' ;192 bytes (1536.0 bits ==> 1536 bits)
define BYTES19200_16 '173' ;173 bytes (1382.4 bits ==> 1384 bits)
;:::28.8
define BYTES19200_24 '116' ;116 bytes (921.6 bits ==> 928 bits)
;:::28.8
define BYTES19200_32 '87' ;87 bytes (691.2 bits ==> 696 bits)
define BYTES19200_48 '58' ;58 bytes (460.8 bits ==> 464 bits)

;define sampling rate table of ISO MUSICAM frame header codes
SAMPLERATES macro
sampling
;:::28.8
;:::28.8
if SAMPLE_RATE_PAIR==SAMPLE_16K_AND_24K
dc SAMPLINGRATE_16 ;old CCS CDQ1000 sampling at 14.4 K
dc SAMPLE_ID_BIT_HIGH ;old CCS CDQ1000 header sampling id bit
dc MAXSUBBANDS_CCS ;old CCS CDQ1000 max sub-bands 1 channel

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 51 -

```

dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 1 channel
dc      SAMPLINGRATE_16     ;old CCS CDQ1000 sampling at 14.4 K
dc      SAMPLE_ID_BIT_HIGH   ;old CCS CDQ1000 header sampling id bit
dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 1 channel
dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 2 channel
dc      MAXCRITBANDS_16      ;number of critical bands at 14.4 K
dc      NMSKFREQS_16         ;num freqs used for coding at 14.4 K
dc      SAMPLINGRATE_16     ;old CCS CDQ1000 sampling at 14.4 K
dc      SAMPLE_ID_BIT_HIGH   ;old CCS CDQ1000 header sampling id bit
dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 1 channel
dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 2 channel
dc      SAMPLINGRATE_16     ;old CCS CDQ1000 sampling at 14.4 K
dc      SAMPLE_ID_BIT_HIGH   ;old CCS CDQ1000 header sampling id bit
dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 1 channel
dc      MAXSUBBANDS_CCS      ;old CCS CDQ1000 max sub-bands 2 channel
dc      MAXCRITBANDS_16      ;number of critical bands at 14.4 K
dc      NMSKFREQS_16         ;num freqs used for coding at 14.4 K
;:::28.8
;:::28.8      endif

      endm

;define framing bit rate table

BITRATES      macro

bitrates

;:::28.8      if SAMPLE_RATE_PAIR==SAMPLE_16K_AND_24K
;:::28.8
;entry for code 0      RATE_LO      ;framing bit rate of 28.8 Kbits
dc      BITRATE_56      ;ISO frame header code for 28.8 Kbits
dc      BITRATE_56      ;ISO frame header code for 28.8 Kbits
dc      OUTM56_16      ;num 24 bit wds 28.8 Kbit frame @ 14.4 K sample
dc      OUTB56_16      ;num bits 28.8 Kbit frame @ 14.4 K sample
dc      BITRATE_56      ;ISO frame header code for 28.8 Kbits
dc      BITRATE_56      ;ISO frame header code for 28.8 Kbits
dc      OUTM56_16      ;num 24 bit wds 28.8 Kbit frame @ 14.4 K sample
dc      OUTB56_16      ;num bits 28.8 Kbit frame @ 14.4 K sample

;entry for code 1      RATE_HI      ;framing bit rate of 28.8 Kbits
dc      BITRATE_64      ;ISO frame header code for 28.8 Kbits
dc      BITRATE_64      ;ISO frame header code for 28.8 Kbits
dc      OUTM64_16      ;num 24 bit wds 28.8 Kbit frame @ 14.4 K sample
dc      OUTB64_16      ;num bits 28.8 Kbit frame @ 14.4 K sample
dc      BITRATE_64      ;ISO frame header code for 28.8 Kbits
dc      BITRATE_64      ;ISO frame header code for 28.8 Kbits
dc      OUTM64_16      ;num 24 bit wds 28.8 Kbit frame @ 14.4 K sample
dc      OUTB64_16      ;num bits 28.8 Kbit frame @ 14.4 K sample
;:::28.8
;:::28.8      endif

      endm

;define bit allocation bandwidth tables

BANDWIDTHS      macro

bndwtbl

;:::28.8      if SAMPLE_RATE_PAIR==SAMPLE_16K_AND_24K

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 52 -

```

;...28.8
; KBit rates low/high @ 14400 sampling

dc      USEDSUBBANDS_00_16 ; rate low code 00: mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_01_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_10_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_11_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation

dc      USEDSUBBANDS_00_16 ; rate high code 01: mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_01_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_10_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_11_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation

; KBit rates low/high @ 14400 sampling

dc      USEDSUBBANDS_00_16 ; rate low code 00: mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_01_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_10_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_11_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation

dc      USEDSUBBANDS_00_16 ; rate high code 01: mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_01_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_10_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation
dc      USEDSUBBANDS_11_16 ; mono band-width
dc      LIMITSUBBANDS      ; subbands requiring 1 allocation

;...28.8
;...28.8      endif

endm

;define ancillary data baud rate table of clock values and byte counts
BAUDCLK      macro

baudclk
;...28.8      if SAMPLE_RATE_PAIR==SAMPLE_16K_AND_24K
;...28.8
dc      M_SCCR300          ;set clock for 300 data baud rate      (0)
dc      BYTES300_16        ;tol check of bytecnt @ sample 14.4 K
dc      BYTES300_16        ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR1200         ;set clock for 1200 data baud rate      (1)
dc      BYTES1200_16       ;tol check of bytecnt @ sample 14.4 K
dc      BYTES1200_16       ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR2400         ;set clock for 2400 data baud rate      (2)
dc      BYTES2400_16       ;tol check of bytecnt @ sample 14.4 K

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 53 -

```

dc      BYTES2400_16      ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR3600        ;set clock for 3600 data baud rate (3)
dc      BYTES3600_16      ;tol check of bytecnt @ sample 14.4 K
dc      BYTES3600_16      ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR4800        ;set clock for 4800 data baud rate (4)
dc      BYTES4800_16      ;tol check of bytecnt @ sample 14.4 K
dc      BYTES4800_16      ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR38400       ;set clock for 38400 data baud rate (5)
dc      BYTES38400_16     ;tol check of bytecnt @ sample 14.4 K
dc      BYTES38400_16     ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR9600        ;set clock for 9600 data baud rate (6)
dc      BYTES9600_16      ;tol check of bytecnt @ sample 14.4 K
dc      BYTES9600_16      ;tol check of bytecnt @ sample 14.4 K
dc      M_SCCR19200       ;set clock for 19200 data baud rate (7)
dc      BYTES19200_16     ;tol check of bytecnt @ sample 14.4 K
dc      BYTES19200_16     ;tol check of bytecnt @ sample 14.4 K
;!!!28.8
;!!!28.8      endif

      endm

;define MICRO decoder Auto Select MUSICAM frame sizes to determine if:
;      input data is MUSICAM frames vs G722 data
;      what is the framing bit rate and sampling rate

AUTOFRAME      macro

autotbl
;!!!28.8      if SAMPLE_RATE_PAIR==SAMPLE_16K_AND_24K
;!!!28.8
dc      OUTM56_16      ;96 words in 28.8 Kbit frame @ 14.4 KHz
dc      OUTM64_16      ;96 words in 28.8 Kbit frame @ 14.4 KHz
dc      OUTM56_16      ;96 words in 28.8 Kbit frame @ 14.4 KHz
dc      OUTM64_16      ;96 words in 28.8 Kbit frame @ 14.4 KHz
;!!!28.8
;!!!28.8      endif

      endm

.....
end of box_ctl.asm
.....

list

```



- 54 -

```

op:      fc

(c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.
; ADGCST\dcframe.asm: u_psych parameter for findrms vs checksub
; title 'PCM data thru XPSYCHO and XCODE'
; multiple mono channels
; This routine receives a buffer of PCM data and builds a stand alone
; single channel mono frame for multiple mono channel devices

cn entry
    r0 = address of the input PCM buffer
    r1 = address of the coded frame buffer

cn exit
    a = destroyed
    b = destroyed
    y0 = destroyed
    y1 = destroyed
    r0 = destroyed
    r1 = destroyed
    r4 = destroyed
    n4 = destroyed

    include 'def.asm'

    section highmisc
    xdef  ntonals
    xdef  nmasker

    org  xhe:
stdcframe_xhe
ntonals ds 1 ;number of tonals in tonal structure
nmasker ds 1 ;number of maskers in masker structure

enddcframe_xhe
endsec

    section ytables
    xdef  rngtbl

    org  yhe:
stdcframe_ytbl
rngtbl
    dc 2,3,6,6,12,12,12,12 ;table for searching for tonals

enddcframe_ytbl
endsec

    org  phe:

dcframe
;:::dbg

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 55 -

```

        rts      ;jmp      <top          ;!!! debug :if using stored frames buffer
        jmp      <_xcode_
        jmp      <_polya_
!!!dbg

```

```

.....
.....
.....      Start XPSYCHO      .....
.....
.....

```

```

; Now get the position to read the fft data from
; This buffer is offset from the polyphase filter to account for the
; delay through the filter

```

```

        move     #PCMSIZE-1,m0          ;set to a mod buffer
        move     y:<polyst,r0           ;get input pcm buffer address
        move     #(256-64),n0          ;back up to position fft
        move     #hbuf,r1              ;get hanning output buffer address
        move     (r0)-n0

        jsr      <hanning               ;apply a hanning window
        move     y:<linear,m0           ;restore r0 to linear buffer

        jsr      <fft                   ;fft the data

        move     #fftbuf,r0             ;real part of fft
        move     #fftbuf,r4            ;imaginary part of fft
        move     #power,r1              ;power array
        jsr      <logpow                 ;compute power of fft data

        move     #power,r0              ;power array
        move     #SBMaxDb,r1            ;maximum in each sub-band (slb)
        jsr      <findmaxi               ;find max power in a sub-band

        move     #power,r1              ;power array
        move     #Tonals,r2             ;tonal array
        move     #rngtbl,r4             ;range table for tonal search
        jsr      <findtona               ;find tonals
        move     r3,x:ntonals            ;save number of tonals

        move     #power,r1              ;power array
        move     #Tonals,r2             ;tonal array
        move     #rngtbl,r4             ;range table for tonal search
        jsr      <zeropowe               ;zero power around tonals

        move     #power,r1              ;power array
        move     #NoisePwr,r2           ;address of the noise array
        jsr      <findnois               ;find the noise

        move     #Maskers,r3            ;address of the masker structure
        move     #NoisePwr,r2           ;address of the noise array
        move     #Tonals,r1             ;address of the Tonals structure
        move     x:ntonals,x0           ;# of tonals in Tonals structure
        jsr      <mergemas               ;merge the maskers
        move     b,x:nmasker            ;save # of maskers

        move     #Maskers,r0            ;get address of the Masker structure
        move     x:nmasker,b            ;number of maskers in masker structure
        jsr      <finddbma               ;find the db value of maskers

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL 

- 56 -

```

        move    #Maskers,r0          ;get address of the Masker structure
        jsr     <pruneclo            ;prune close maskers

        move    #Maskers,r0          ;get address of the Masker structure
        move    x:nmasker,b          ;number of maskers in masker structure
        jsr     <prunequi            ;prune quiet maskers

        move    #Maskers,r0          ;get address of the Masker structure
        move    x:nmasker,b          ;number of maskers in masker structure
        jsr     <prunemas            ;prune masked maskers

        move    #Tonals,r0           ;address of the Tonals structure
        move    x:ntonals,x0         ;# of tonals in Tonals structure
        move    #Alisng,r1           ;destination buffer address
        jsr     <findalis            ;find alising components
        move    b,x:nalias

        move    #Maskers,r4          ;get address of the Masker structure
        move    #GlbMsk,r1           ;address of global masking threshold
        jsr     <QCalcGlc            ;calculate global masking threshold

_polya_
; polyphase filter the input data

        move    y:<polyst,r0         ;get polyana start address
        move    #PCMSIZE-1,m0        ;set as a mod buffer
        move    #PlAnal,r5           ;set start of the sub-band output buffer
        jsr     <polyanal            ;poly analyze the data
        move    y:<linear,m0         ;restore to linear ctl

; develop the scale factors
; initialize the table of scale factors to minimum amplitude (63 ==> 0 ampl)

        move    #SBndSKF,r0          ;addr of sub-band scale factors
        move    #63,n4

        do      #NUMSUBBANDS*NPERGROUP, _init_00
        move    n4,x:(r0)+           ;get value to store shared memory

_init_00
        move    #PlAnal,r0           ;addr of poly analyzed data
        move    #SBndSKF,r1          ;addr of sub-band scale factors
        jsr     <findskf            ;find scale factors

; develop the SBits for scale factors

        move    #SBndSKF,r0          ;addr of sub-band scale factors
        move    #SBits,r1            ;addr of sub-band sbits
        jsr     <pickskf            ;pick the best scale factors

_xcode_
.....
.....
..... Start XCODE .....
.....
.....

```



- 57 -

```

; determine which method to use to determine the sub-band maximum values
        move    y:u_psych,a        ;get use findrms.asm rtn parameter
        move    #.5,x1             ;if less than .5, use checksub.asm rtn
        cmp     x1,a               ;see if parameter less than .5
        jlt     <_do_checksub      ;if less, use checksub.asm rtn

; use RMS for maximum level for the sub-band
        move    #PlAnal,r0         ;addr of poly analyzed data
        move    #SBMaxDb,r1        ;addr of sub-band max
        jsr     <findrms           ;find max in a subband
        jmp     <_set_min_mask     ;go to set minimum masking level

; _do_checksub
; set correct maximum level for the channel
        move    #SBndSKF,r0        ;addr of sub-band scale factors
        move    #SBMaxDb,r1        ;addr of sub-band max
        jsr     <checksub          ;find max in a subband

; _set_min_mask
; set minimum masking level in each sub-band
        move    #GlbMsk,r0         ;channel global masking threshold
        move    #MinMskDb,r1       ;minimum masking per subband (slb)
        jsr     <findminm         ;find min masking

; set minimum masking level in each sub-band: left channel then right channel
        move    x:nalias,a         ;number of aliazer's
        move    #Alisng,r0         ;aliasing structure
        move    #SBMaxDb,r1        ;max in each sub-band (slb)
        jsr     <findmaxs         ;find the maximum signal

; set number of fixed bits required, and the number of available bits for audio
        jsr     <bitpool

        move    x0,y:fixbits       ;save fixed bit count
        move    x1,y:audbits       ;save bit count available for alloc

; allocate the bits in the frame by subband
        move    #SBits,r0          ;scale factors
        move    #MinMskDb,r1       ;minimum masking per sub-band (slb)
        move    #SBMaxDb,r2       ;maximum in each sub-band (slb)
        move    #SBPos,r4         ;sub-band position
        move    #SBindx,r5        ;sub-band indices
        jsr     <bitalloc          ;allocate the bits

; code the channel audio frame
        jsr     <codeframe

        rts

```

SUBSTITUTE SHEET (RULE 26)

RAD ORIGINAL



- 58 -

```

op:    fc
;
; (c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \RMICRO\getbal.asm
;
; title 'Get bit allocations'
;
; This routine is used to get the bit allocations of each of the sub-bands.
; It is from the ISO standard.
; sub-band 0 - 10 use 4 bits (11 * 4 = 44 bits)
; sub-band 11 - 22 use 3 bits (12 * 3 = 36 bits)
; sub-band 23 - 26 use 2 bits ( 4 * 2 =  8 bits)
;                                     ( total = 88 bits)
;
; on entry
;   r0 = address of bit allocation array for both left and right channels
;
;   r6 = current offset in the input array
;   n6 = base address of the input array
;   y:<maxsubs = MAXSUBBANDS at sampling rate and bit rate
;   y:sc = shift count of current input word
;   y:frmttype = full stereo, joint stereo or mono
;   y:sbound = joint stereo sub-band intensity bound
;   x:crclbits = accumulator of bits covered by CRC-16 routine
;                                     (bit allocation bits are accumulated)
;
; on exit
;   r6 = updated
;   y:sc = updated
;
;   a = destroyed
;   b = destroyed
;   x0 = destroyed
;   x1 = destroyed
;   y0 = destroyed
;   y1 = destroyed
;   r0 = destroyed
;   r1 = destroyed
;   r2 = destroyed
;   r4 = destroyed
;   n4 = destroyed
;
; include 'def.asm'
;
; .DGCST:
; section highmisc
; xdef  masktbl
; xdef  tbl
;
; org    yhe:
; stgetbal_yhe
;
; masktbl
; dc      5000000 ;place holder in mask table
; dc      5000001 ;mask table for 1 bit getvalue
; dc      5000003 ;mask table for 2 bit getvalue
; dc      5000007 ;mask table for 3 bit getvalue
; dc      500000f ;mask table for 4 bit getvalue

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 59 -

```

;;      dc      $00001f      ;mask table for 5 bit getvalue
;;      dc      $00003f      ;mask table for 6 bit getvalue
;;      dc      $00007f      ;mask table for 7 bit getvalue
;;      dc      $0000ff      ;mask table for 8 bit getvalue
;;      dc      $0001ff      ;mask table for 9 bit getvalue
;;      dc      $0003ff      ;mask table for 10 bit getvalue
;;      dc      $0007ff      ;mask table for 11 bit getvalue
;;      dc      $000fff      ;mask table for 12 bit getvalue
;;      dc      $001fff      ;mask table for 13 bit getvalue
;;      dc      $003fff      ;mask table for 14 bit getvalue
;;      dc      $007fff      ;mask table for 15 bit getvalue
;;      dc      $00ffff      ;mask table for 16 bit getvalue
;;
;;;define data size table for the getvalue routine to extract data
;;
;;tbl
;;      dc      $000000      ;bits = 0, place holder
;;      dc      $000001      ;shift left 01 bits
;;      dc      $000002      ;shift left 02 bits
;;      dc      $000004      ;shift left 03 bits
;;      dc      $000008      ;shift left 04 bits
;;      dc      $000010      ;shift left 05 bits
;;      dc      $000020      ;shift left 06 bits
;;      dc      $000040      ;shift left 07 bits
;;      dc      $000080      ;shift left 08 bits
;;      dc      $000100      ;shift left 09 bits
;;      dc      $000200      ;shift left 10 bits
;;      dc      $000400      ;shift left 11 bits
;;      dc      $000800      ;shift left 12 bits
;;      dc      $001000      ;shift left 13 bits
;;      dc      $002000      ;shift left 14 bits
;;      dc      $004000      ;shift left 15 bits
;;      dc      $008000      ;shift left 16 bits
;;
;;endgetbal_yhe
;;      endsec

      section highmisc
      xdef      skftbl
      xdef      skftbl_1
      xdef      skftbl_2
      xdef      skftbl_3

      org      xhe:
stgetbal_xhe

;address of BAL's bit table as per Allowed table selected
skftbl ds      1

;These tables is the number of bits used by the scale factor in each sub-band
; High sampling rates with higher bit rate framing
skftbl_1
      dc      4      ;sub-band 0
      dc      4      ;sub-band 1
      dc      4      ;sub-band 2
      dc      4      ;sub-band 3

```



- 60 -

dc	4	;sub-band 4
dc	4	;sub-band 5
dc	4	;sub-band 6
dc	4	;sub-band 7
dc	4	;sub-band 8
dc	4	;sub-band 9
dc	4	;sub-band 10
dc	3	;sub-band 11
dc	3	;sub-band 12
dc	3	;sub-band 13
dc	3	;sub-band 14
dc	3	;sub-band 15
dc	3	;sub-band 16
dc	3	;sub-band 17
dc	3	;sub-band 18
dc	3	;sub-band 19
dc	3	;sub-band 20
dc	3	;sub-band 21
dc	3	;sub-band 22
dc	2	;sub-band 23
dc	2	;sub-band 24
dc	2	;sub-band 25
dc	2	;sub-band 26
;end table 3-B.2a		
dc	2	;sub-band 27
dc	2	;sub-band 28
dc	2	;sub-band 29
;end table 3-B.2b		
dc	2	;sub-band 30
dc	2	;sub-band 31

; High sampling rates with lower bit rate framing

skftbl_2		
dc	4	;sub-band 0
dc	4	;sub-band 1
dc	3	;sub-band 2
dc	3	;sub-band 3
dc	3	;sub-band 4
dc	3	;sub-band 5
dc	3	;sub-band 6
dc	3	;sub-band 7
;end table 3-B.2c		
dc	3	;sub-band 8
dc	3	;sub-band 9
dc	3	;sub-band 10
dc	3	;sub-band 11
;end table 3-B.2d		
dc	3	;sub-band 12
dc	3	;sub-band 13
dc	3	;sub-band 14
dc	3	;sub-band 15
dc	3	;sub-band 16
dc	3	;sub-band 17
dc	3	;sub-band 18
dc	3	;sub-band 19
dc	3	;sub-band 20

- 61 -

```

dc      3      ;sub-band 21
dc      3      ;sub-band 22
dc      3      ;sub-band 23
dc      3      ;sub-band 24
dc      3      ;sub-band 25
dc      3      ;sub-band 26
dc      3      ;sub-band 27
dc      3      ;sub-band 28
dc      3      ;sub-band 29
dc      3      ;sub-band 30
dc      3      ;sub-band 31

; Low sampling rates
skf:bl_3
dc      4      ;sub-band 0
dc      4      ;sub-band 1
dc      4      ;sub-band 2
dc      4      ;sub-band 3

dc      3      ;sub-band 4
dc      3      ;sub-band 5
dc      3      ;sub-band 6
dc      3      ;sub-band 7
dc      3      ;sub-band 8
dc      3      ;sub-band 9
dc      3      ;sub-band 10

dc      2      ;sub-band 11
dc      2      ;sub-band 12
dc      2      ;sub-band 13
dc      2      ;sub-band 14
dc      2      ;sub-band 15
dc      2      ;sub-band 16
dc      2      ;sub-band 17
dc      2      ;sub-band 18
dc      2      ;sub-band 19
dc      2      ;sub-band 20
dc      2      ;sub-band 21
dc      2      ;sub-band 22
dc      2      ;sub-band 23
dc      2      ;sub-band 24
dc      2      ;sub-band 25
dc      2      ;sub-band 26
dc      2      ;sub-band 27
dc      2      ;sub-band 28
dc      2      ;sub-band 29
;end table 3-B.1
dc      2      ;sub-band 30
dc      2      ;sub-band 31

```

```

endgetbal_xhe
endsec

```

```

org phe:

```

```

;initialize:
; a. ri with start of subband allocation table of bits in frame per sub-band
; b. no offset for right channel sub-band bit allocation values:
; left channel from 0 to (NUMSUBBANDS - 1)

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 62 -

right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)
 c. r3 set with joint stereo sub-band boundary for stereo intensity:
 4 (4-31), 8 (8-31), 12 (12-31) or 16 (16-31)

```
getbal      move    x:skftbl,r1
            move    #masktbl,r2
            move    #NUMSUBBANDS,n0      ;offset for right channel
            move    y:<sibound,r3        ;decr stereo intens sub-band ctr
            move    x:crcbits,r5        ;get CRC-16 bit counter

;loop through the sub-bands extracting the left and right (if applicable)
;bit allocation index values (y:<maxsubs = fixed count of sub-bands framed):
a. for current sub-band get the number of bits for allocation index value
   and increment address of the next sub-band bit count
b. get the bit allocation for the left channel always
c. b register isolate the type of frame: full stereo, joint stereo or mono
d. y0 holds the mono frame type code for testing
e. y1 holds the joint stereo frame type code for testing
f. see if the frame type is joint stereo and just in case, move the
   current stereo intensity sub-band boundary counter value for testing
g. if not joint stereo, see if this is a mono frame type
h. if it is joint stereo:
   1. test if the boundary counter has reached zero, and just in case it has,
      restore the left channel bit allocation value to the a1 register
   2. if the counter is zero, go to copy left channel into the right channel
   3. if not, go to extract the full stereo right channel allocation value

do          y:<maxsubs,_getb_40
move        x:(r1)+,n4      ;get # of bits to read
move        n4,n2          ;get hi order bit mask index
move        n4,n5          ;to accumulate CRC-16 bits
jsr         <getvalue      ;get a left chan bit allocation
move        y:(r2+n2),x1    ;mask for high order one's
move        (r5)+,n5        ;accum bits for CRC-16 rtn
and         x1,a          y:<frmttype,b ;mask off high order one's
                                     ; & set for frame type compare
move        a1,x:(r0)      ;set left channel
move        #>MONO,y0      ;ck for no right channel
move        #>JOINT_STEREO,y1 ;ck for intensity sub-band
cmp         y1,b          r3,a  ;check for stereo intensity
jne         <_getb_10      ;if not, see if mono
tst         a          x:(r0),a1 ;reached bound, restore left val
jeq         <_getb_30      ;yes, left val to right val
move        (r3)-         ;no, decr intens sub-band ctr
jmp         <_getb_20      ;and retrieve right chan value

;test for a mono type of frame and just in case it is, set a1 to zero
;for insertion into the right channel for consistency
;if it is mono, go to move the right channel value
;otherwise, fall through to full stereo
```

```
_getb_10    cmp         y0,b          #0,a1      ;if mono, insert 0 for right
            jeq         <_getb_30

;full stereo, extract the right channel bit allocation value
_getb_20    jsr         <getvalue      ;get a right chan bit allocation
```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 63 -

```

        move    y:(r2+n2),x1          ;mask for high order one's
        move    (r5)+n5               ;accum bits for CRC-16 rtn
        and     x1,a                  ;mask off high order one's

;insert the right channel value (n0 offset)
;increment for the next sub-band

_getb_30
        move    a1,x:(r0+n0)          ;right channel sub-band alloc
        move    (r0)+                ;incr for next sub-band

_getb_40
; Fill the unused sub-bands with 0 bit allocation
; This allows getdata to process these sub-bands normally and insert 0
; data in them.

        clr     a                     #>NUMSUBBANDS,b
        move    y:<maxsubs,x0         ;current MAXSUBBANDS
        sub     x0,b                  ;equals unused sub-bands
        do      b,_getb_50
        move    a,x:(r0+n0)          ;right channel
        move    a,x:(r0)+            ;left chan & incr for next

_getb_50
        move    r5,x:crcbits          ;store updated CRC-16 bit counter

        rts

```



- 64 -

```

opt fc,cex,mex
; (c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \DGCST\getdata.asm: moves to high P-Memory
; title 'Get the Data'
; This routine sets the data in the output buffer
; on entry
; r3 = address of left & right channel SubBandIndex array (x memory)
; r2 = address of left & right channel SubBandSKFs array (x memory)
; r1 = addr of buffer for a set of left and right channel recovered data:
;      (192 samples: one group of 3 samples, 32 subbands, 2 channels)
; y:<maxsubs = MAXSUBBANDS at sampling rate and bit rate
; y:AllwAdd = address of the proper Allowed table at sample/bit rates
; y:frmtype = whether full stereo, joint stereo or mon frame
; y:sibound = if joint stereo, sub-band boundary for stereo intensity
; shared memory for rsynth
; on exit
; a = destroyed
; b = destroyed
; x0 = destroyed
; x1 = destroyed
; y0 = destroyed
; y1 = destroyed
; r0 = destroyed
; r1 = destroyed
; r2 = destroyed
; r3 = destroyed
; r4 = destroyed
; r5 = destroyed
; n0 = destroyed
; n1 = destroyed
; n2 = destroyed
; n3 = destroyed
; n4 = destroyed
; n5 = destroyed

include 'def.asm'
include '..\rmicro\getvalue.mac'

section highmisc
xdef  NBits
xdef  CC
xdef  DD
xdef  packmax
xdef  packrpl

org    xhe:
stgetdata_xhe

NBits
dc      0
dc      2
dc      3
dc      3
dc      4
;position = 0, place holder
;position = 1
;position = 2
;position = 3
;position = 4

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 65 -

```

dc      4      ;position = 5
dc      5      ;position = 6
dc      6      ;position = 7
dc      7      ;position = 8
dc      8      ;position = 9
dc      9      ;position = 10
dc     10      ;position = 11
dc     11      ;position = 12
dc     12      ;position = 13
dc     13      ;position = 14
dc     14      ;position = 15
dc     15      ;position = 16
dc     16      ;position = 17

```

CC

```

dc      0      ;position 0, place holder
dc     $555555 ; 4.0/(3.0*2.0) position 1 */
dc     $666666 ; 8.0/(5.0*2.0) position 2 */
dc     $492492 ; 8.0/(7.0*2.0) position 3 */
dc     $71C71C ; 16.0/(9.0*2.0) position 4 */
dc     $444444 ; 16.0/(15.0*2.0) position 5 */
dc     $421084 ; 32.0/(31.0*2.0) position 6 */
dc     $410410 ; 64.0/(63.0*2.0) position 7 */
dc     $408102 ; 128.0/(127.0*2.0) position 8 */
dc     $404040 ; 256.0/(255.0*2.0) position 9 */
dc     $402010 ; 512.0/(511.0*2.0) position 10 */
dc     $401004 ; 1024.0/(1023.0*2.0) position 11 */
dc     $400801 ; 2048.0/(2047.0*2.0) position 12 */
dc     $400400 ; 4096.0/(4095.0*2.0) position 13 */
dc     $400200 ; 8192.0/(8191.0*2.0) position 14 */
dc     $400100 ; 16384.0/(16383.0*2.0) position 15 */
dc     $400080 ; 32768.0/(32767.0*2.0) position 16 */
dc     $400040 ; 65536.0/(65535.0*2.0) position 17 */

```

DD

```

dc     $000000 ; position 0, place holder
dc     $c00000 ; position 1, .5000000-1.0 */
dc     $c00000 ; position 2, .5000000-1.0 */
dc     $a00000 ; position 3, .2500000-1.0 */
dc     $c00000 ; position 4, .5000000-1.0 */
dc     $900000 ; position 5, .1250000-1.0 */
dc     $880000 ; position 6, .0625000-1.0 */
dc     $840000 ; position 7, .0312500-1.0 */
dc     $820000 ; position 8, .0015625-1.0 */
dc     $810000 ; position 9, .0007812-1.0 */
dc     $808000 ; position 10, .0003906-1.0 */
dc     $804000 ; position 11, .0001953-1.0 */
dc     $802000 ; position 12, .0000976-1.0 */
dc     $801000 ; position 13, .0000488-1.0 */
dc     $800800 ; position 14, .0000244-1.0 */
dc     $800400 ; position 15, .0000122-1.0 */
dc     $800200 ; position 16, .0000061-1.0 */
dc     $800100 ; position 17, .0000030-1.0 */

```

;check for bit errors in packed positions: 1, 2, 3 and 4

position	STANDARD ISO		CCS COMPRESSED	
	max replacement value	value	max replacement value	value
1	26	13	14	7
2	124	62	62	31

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 66 -

```

      3      ---      ---      438      219
      4      728      364      ---      ---

packmax dc      1
packrpl dc      1

endgetdata_xhe
endsec

      section lowmisc
      xdef      av
      xdef      bv
      xdef      cv
      xdef      bandcnt
      xdef      block
      xdef      svereg
      xdef      dvalue, cvalue

      org      yli:
stgetdata_yli

av      ds      1      ;A value after uppacking
bv      ds      1      ;B value after uppacking
cv      ds      1      ;C value after uppacking
bandcnt ds      1      ;incr sub-band for stereo intensity
block   ds      1      ;block no 0:0-3, 1:4-7, 2:8-11
svereg  ds      1      ;save a register value
dvalue  ds      1      ;hold current DValue
cvalue  ds      1      ;hold current CValue

endgetdata_yli
endsec

      section highmisc
      xdef      ivdata
      xdef      ASMDadd
      xdef      SKFaddr
      xdef      INXaddr
      xdef      AllwAdd
      xdef      Allow
      xdef      getdataN4Save
      xdef      bereich
      xdef      shftbl

      org      yhe:
stgetdata_yhe

ivdata  ds      1      ;left & right channel recovered data
ASMDadd ds      1      ;A start addr shared mem for samples
SKFaddr ds      1      ;starting addr for SKF's
INXaddr ds      1      ;starting addr for SBIndx's
AllwAdd ds      1      ;save addr of applicable Allowed table
Allow   ds      1      ;current address in Allowed for sb
getdataN4Save ds      1

      include '...\common\bereich.asm'

shftbl  dc      $000000      ;bits = 0, place holder

```

- 67 -

```

dc      $400000      ;bits = 1, shift left 23 bits
dc      $200000      ;bits = 2, shift left 22 bits
dc      $100000      ;bits = 3, shift left 21 bits
dc      $080000      ;bits = 4, shift left 20 bits
dc      $040000      ;bits = 5, shift left 19 bits
dc      $020000      ;bits = 6, shift left 18 bits
dc      $010000      ;bits = 7, shift left 17 bits
dc      $008000      ;bits = 8, shift left 16 bits
dc      $004000      ;bits = 9, shift left 15 bits
dc      $002000      ;bits = 10, shift left 14 bits
dc      $001000      ;bits = 11, shift left 13 bits
dc      $000800      ;bits = 12, shift left 12 bits
dc      $000400      ;bits = 13, shift left 11 bits
dc      $000200      ;bits = 14, shift left 10 bits
dc      $000100      ;bits = 15, shift left 09 bits
dc      $000080      ;bits = 16, shift left 08 bits

endgetdata_yhe
endsec

org      phe:

getdata
move     r2,y:SKFAddr      ;save start address
move     r3,y:INXAddr      ;save start address
move     r1,y:ASMDAddr     ;save start addr ivquant values

move     #0,r0              ;start group number

;loop through the 12 groups of 3 samples per sub-band per channel
; advancing through 36 samples
; set-up for the group:
; 1. set starting address for inverse quantized values
; 2. reset the starting address of the Allowed sub-band bits
; 3. determine the SKF factor grouping
; 4. set up for joint stereo sub-band intensity boundary checking

do        #NUMPERSUBBAND,_getd_90

; set up for next group of samples

move     y:ASMDAddr,r1      ;reset start recover data addr
move     r1,y:ivdata        ;init recovered data curr addr
move     y:INXAddr,r3       ;reset SBindx ptr
move     y:SKFAddr,r2       ;reset start SKF address
move     y:AllwAddr,r5      ;reset address of allowed
move     r5,y:Allow         ;and save

;set which block of SKFs (scale factor indices):
; 0 for group of 4 samples 0-3
; 1 for group of 4 samples 4-7
; 2 for group of 4 samples 8-11

move     r0,x0              ;curr group to test
move     #>4,b              ;block [0] groups 0 - 3
cmp      x0,b               ;block [0] groups 0 - 3
jgt      <_getd_06

move     #>8,b              ;block [1] groups 4 - 7
cmp      x0,b               ;block [1] groups 4 - 7

```

- 68 -

```

        jgt      <_getd_06
        move     #>2,y1          ;block [2] groups 8 - 11
        _getd_06
        move     (r0)+          ;increment the group number
        move     y1,y:<block    ;save which block[0, 1 or 2]

;set-up for joint stereo sub-band intensity control
        move     y:<sbound,n0    ;joint stereo intensity sub-band
        move     n0,y:<bndcnt    ;bound sub-band decremented cnt
        bclr     #JOINT_at_SB_BOUND,y:<ctlflgs ;clear reached intensity sub-band

;process this collection of three samples per sub-band per channel
        do       #NUMSUBBANDS,_getd_80
        move     y:ivdata,r1    ;left channel block 1st
        move     #0,n3          ;left channel SBindx values
        bclr     #LEFT_vs_RIGHT,y:<ctlflgs ;indicate working on left chan
        move     y:<block,n2    ;which block of SKFs

;process left channel and then right channel for current sub-band
        do       #NUMCHANNELS,_getd_75
        move     #NUMSUBBANDS,n1 ;spaced by number of subbands
        move     x:(r3+n3),n5    ;SubBandIndex[SubBand]
        move     y:Allow,r5     ;get the address of Allowed[SB]
        move     #DD,r4        ;address of the D table
        move     x:(r5+n5),n5    ;get position for the subband
        move     n5,a          ;save the position

        tst      a,n5,n4       ;check position == 0 AND
                                ; set position for DValue fetch
                                ;not transmitted
        jeq      <_getd_60

        move     #CC,r5         ;address of the C table
        move     x:(r4+n4),x1    ;DValue
        move     x:(r5+n5),x0    ;CValue
        move     x1,y:<dvalue    ;save DValue
        move     x0,y:<cvalue    ;save CValue

        move     #NBits,r5      ;address of NBits array
        move     #>1,y1         ;to test for packed pos : below
        move     x:(r5+n5),n4    ;nbits

        move     x:(r2+n2),n5    ;SKFIndex[SubBand][block]
        move     #bereich,r5     ;SKF table address

;now, if doing the left channel, continue with extracting data
;otherwise, check for joint stereo and the intensity bound of sub-band
;if right channel joint stereo sub-band intensity boundary reached.
; inverse quantize the saved raw values extracted for the left channel
;otherwise extract the true right channel stereo values for inverse quantizing
        jclr     #LEFT_vs_RIGHT,y:<ctlflgs,_getd_10 ;clear if doing on left chan
        jset     #JOINT_at_SB_BOUND,y:<ctlflgs,_getd_50 ;reached bound, do right

```



- 69 -

_getd_13

; a. set up for extracting the data values
 ; b. test the position for packed types (positions: 1, 2, 3 or 4)

```

move    #tbl,r4          ;get shift table address
move    n4,n0            ;save nbits
move    y:<sc,b           ;get the shift count
move    y:<curwd,y0       ;get current frame word
cmp      y1,a    #>2,y1   ;check position == 1
jeq      <_getd_20        ;handle pos 1 with 3 packed values
cmp      y1,a    #>4,y1   ;check position == 2
jeq      <_getd_30        ;handle pos 2 with 3 packed values
cmp      y1,a    #>3,y1   ;check position == 4
jeq      <_getd_40        ;handle pos 4 with 3 packed values
cmp      y1,a           ;check position == 3, and if not,
jne      <_getd_12        ;handle all other pos as unpacked

```

; for position 3:
 ; if compressed mode, handle allocation as a packed value
 ; otherwise, handle as ISO standard unpacked set of 3 values

```

jset     #DECOMPRESS_PACKED,y:<ctrlflgs,_getd_35

```

_getd_12

; not position 1, 2 or 4 so just a regular input of 3 adjacent data values

```

move     y:(r4+n4),x0     ;get shift left multiplier per bit cnt

```

; extract the 1st value and save it in y:<av

```

mpy      x0,y0,a n4,x1    ;shift extracted bits into a1 with
                           ; newly shifted curwd in a0
                           ; & save passed numb bits required
sub      x1,b    a0,y:<curwd ;see if next word need to complete value
                           ; & save newly shifted curwd
move     b,y:<sc          ;save new shift count

```

;let's try a macro

```

jge      <_getd_16
getnextword 10,15

```

_getd_16

```

move     a1,y:<av          ;save 1st for inverse quant

```

; extract the 2nd value and save it in y:<bv

```

move     y:<curwd,y0       ;get current frame word
move     y:(r4+n4),x0     ;get shift left multiplier per bit cnt
mpy      x0,y0,a n4,x1    ;shift extracted bits into a1 with
                           ; newly shifted curwd in a0
                           ; & save passed numb bits required
sub      x1,b    a0,y:<curwd ;see if next word need to complete value
                           ; & save newly shifted curwd
move     b,y:<sc          ;save new shift count

```

;let's try a macro

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 70 -

```

jge      <_getd_18
getnextword 20,25

_getd_18
move     a1,y:<bv           ;save 2nd for inverse quant
; extract the 3rd value and save it in y:<cv

move     y:<curwd,y0        ;get current frame word
move     y:(r4+n4),x0       ;get shift left multiplier per bit cnt
mpy      x0,y0,a n4,x1      ;shift extracted bits into a1 with
                           ; newly shifted curwd in a0
                           ; & save passed numb bits required
sub      x1,b      a0,y:<curwd ;see if next word need to complete value
                           ; & save newly shifted curwd
move     b,y:<sc            ;save new shift count
jsl      <getnextword       ;yes, get rest from next i/p frame word

move     a1,y:<cv           ;save 3rd for inverse quant
jmp      <_getd_50         ;go to do inverse quantizing

; Pos 1: Three adjacent data values are packed into 5 bits.
; Each of the data values are only 2 bits wide.

packed_value = value0 * 9 + value1 * 3 + value2
or
packed_value = 3 * (value0 * 3 + value1) + value2

_getd_20
move     #>26,x0           ;ISO maximum packed value
move     #>13,x1           ;ISO replacement value
move     #MASKUPACK3,n4    ;unpack getvalue mask

; if compressed, switch to compressed mask

jclr     #DECOMPRESS_PACKED,y:<ctlflgs,_getd_21
move     #>14,x0           ;CCS compression maximum packed value
move     #>7,x1            ;CCS compression replacement value
move     #MASKUPACK3X,n4   ;compressed unpack getvalue mask

_getd_21
move     n4,y:<av          ;save in y:<avalue for now
move     #36,n4           ;unpack initial divisor
move     n4,y:<bv          ;save in y:<bvalue for now
move     #9,n4            ;unpack initial multiplier
move     n4,y:<cv          ;save in y:<cvalue for now
move     #12,n4           ;unpack second divisor
move     n4,y:<crcstrt     ;save in y:<crcstrt for now
move     #3,n4            ;unpack second multiplier
move     n4,y:<svereg       ;save in y:<svereg for now
move     #3,n4            ;unpack loop counter
move     n4,y:<not_appl    ;save in y:<not_appl for now
move     #5,n4            ;change to packed values nbits

; if compressed, switch to compressed nbits

jclr     #DECOMPRESS_PACKED,y:<ctlflgs,_getd_22
move     #4,n4            ;change to compress packed values nbits

_getd_22

```

BAD ORIGINAL

SUBSTITUTE SHEET (RULE 26)


```

                                - 71 -
jmp      <_getd_45

; Pos 2: Three adjacent data values are packed into 7 bits.
; Each of the data values are only 3 bits wide.
;
packed_value = value0 * 25 + value1 * 5 + value2
;
; or
packed_value = 5 * (value0 * 5 + value1) + value2

_getd_30
move     #>124,x0                ;ISO maximum packed value
move     #>62,x1                 ;ISO replacement value
move     #MASKUPACK5,n4         ;unpack getvalue mask

; if compressed, switch to compressed mask

jclr     #DECOMPRESS_PACKED,y:<ctlflgs,_getd_31
move     #>62,x0                ;CCS compression maximum packed value
move     #>31,x1                 ;CCS compression replacement value
move     #MASKUPACK5X,n4        ;compressed unpack getvalue mask

_getd_31
move     n4,y:<av                 ;save in y:<avalue for now
move     #200,n4                 ;unpack initial divisor
move     n4,y:<bv                 ;save in y:<bvalue for now
move     #25,n4                 ;unpack initial multiplier
move     n4,y:<cv                 ;save in y:<cvalue for now
move     #40,n4                 ;unpack second divisor
move     n4,y:<crcstrt            ;save in y:<crcstrt for now
move     #5,n4                 ;unpack second multiplier
move     n4,y:<svereg             ;save in y:<svereg for now
move     #4,n4                 ;unpack loop counter
move     n4,y:<not_appl          ;save in y:<not_appl for now
move     #7,n4                 ;change to packed values nbits

; if compressed, switch to compressed nbits

jclr     #DECOMPRESS_PACKED,y:<ctlflgs,_getd_32
move     #6,n4                 ;change to compress packed values nbits

_getd_32
jmp      <_getd_45

; Compressed pos 3:
; Three adjacent data values are packed into 8 bits.
; Each of the data values are only 3 bits wide.
;
packed_value = value0 * 64 + value1 * 8 + value2
;
; or
packed_value = 8 * (value0 * 8 + value1) + value2

_getd_35
move     #>438,x0                ;CCS compression maximum packed value
move     #>219,x1                 ;CCS compression replacement value
move     #MASKUPACK8X,n4        ;unpack getvalue mask
move     n4,y:<av                 ;save in y:<avalue for now
move     #200,n4                 ;unpack initial divisor
move     n4,y:<bv                 ;save in y:<bvalue for now
move     #25,n4                 ;unpack initial multiplier
move     n4,y:<cv                 ;save in y:<cvalue for now

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 72 -

```

move    #40,n4           ;unpack second divisor
move    n4,y:<crcstrt     ;save in y:<crcstrt for now
move    #5,n4            ;unpack second multiplier
move    n4,y:<svereg       ;save in y:<svereg for now
move    #4,n4            ;unpack loop counter
move    n4,y:<not_appl     ;save in y:<not_appl for now
move    #8,n4            ;change to packed values nbits
jmp     <_getd_45

```

; Pos 4: Three adjacent data values are packed into 10 bits.
; Each of the data values are only 4 bits wide.

```

packed_value = value0 * 81 + value1 * 9 + value2
              or
packed_value = 9 * (value0 * 9 + value1) + value2

```

```

_getd_40
move    #>728,x0          ;ISO maximum packed value
move    #>364,x1          ;ISO replacement value
move    #MASKUPACK9,n4    ;unpack getvalue mask
move    n4,y:<av           ;save in y:<avalue for now
move    #1296,n4          ;unpack initial divisor
move    n4,y:<bv           ;save in y:<bvalue for now
move    #81,n4            ;unpack initial multiplier
move    n4,y:<cv           ;save in y:<cvalue for now
move    #144,n4           ;unpack second divisor
move    n4,y:<crcstrt     ;save in y:<crcstrt for now
move    #9,n4             ;unpack second multiplier
move    n4,y:<svereg       ;save in y:<svereg for now
move    #5,n4            ;unpack loop counter
move    n4,y:<not_appl     ;save in y:<not_appl for now
move    #10,n4            ;change to packed values nbits
nop

```

;handle the data value extraction from the frame and unpack for
;either position 1, 2, 3 (if compressed) or 4

```

_getd_45
move    x0,x:packmax      ;save position max packed value
move    x1,x:packrpl      ;save position replacement value
move    y:(r4+n4),x0      ;get shift left multiplier per bit cnt

jclr    #DECOMPRESS_PACKED,y:<ctrlflgs,_getd_46
move    n4,y:getdataN4Save ;save the bit field size

```

```

_getd_46
mpy     x0,y0,a n4,x1     ;shift extracted bits into a1 with
                          ; newly shifted curwd in a0
                          ; & save passed numb bits required
sub     x1,b a0,y:<curwd   ;see if next word need to complete value
                          ; & save newly shifted curwd
move    b,y:<sc            ;save new shift count
jslt    <getnextword      ;yes, get rest from next i/p frame word

move    y:<av,x1           ;unpack getvalue mask
and     x1,a               ;mask off high order one's
move    a1,a               ;clean up

```

;test for a possible bit error that might have caused a value above the
;maximum packed value

- 73 -

;if above maximum, replace with the middle value

```

move    x:packmax,x1      ;get poission max packed dvalue
cmp     x1,a              ;compare retrieved value to max
jle     <_getd_47         ;if not above max value, continue
move    x:packrpl,a       ;since above, replace value

```

_getd_47

```

jclr    #DECOMPRESS PACKED,y:<ctlflgs,_getd_48
move    y:getdataN4Save,r4 ;restore the bit field size
move    a,n4               ;set compressed value for table look up
jsr     <dcompval          ;get the decompressed value for unpack

```

_getd_48

```

jsr     <unpack            ;get 3 parts
move    n0,n4              ;restore nbits

```

;now let's inverse quantize the 3 samples

_getd_50

```

move    #shftbl,r4        ;to left justify in ivquanti
move    y:<av,y0           ;save A value
move    y:(r4+n4),y1       ;get left shift value
tfr     y1,b              ;save left shift in b1
move    y:(r5+n5),b0       ;get C factor

```

;ivquanti 1st value:

```

mpy     y0,y1,a            y:<dvalue,x1 ;1st value: left justify bits
move    a0,a              ; & set DValue
add     x1,a              ;move rslt to correct register
mpy     a1,y0             y:<cvalue,x0 ;Y + D
mpy     x0,y0,a           ; & set CValue
move    a,y1              ;forget sign extension
mpyr    y0,y1,a           ;C * (Y + D)
asl     a                 ; & set up C factor
move    b0,y0             ;rnd scale factor * C * (Y + D)
mpyr    b1,y1             ; & reget left shift value
asl     y:<bv,y0           ;mult by 2 again
move    b1,y1             ; & get B value

```

;ivquanti 2nd value:

```

mpy     y0,y1,a           a,x:(r1)+n1 ;2nd value: left justify bits
move    a0,a              ; & store 1st data value
add     x1,a              ;move rslt to correct register
move    a1,y0             ;Y - D
mpy     x0,y0,a           b0,y0 ;forget sign extension
move    a,y1              ;C * (Y - D)
mpyr    y0,y1,a           b1,y1 ; & reget C factor
asl     a                 ;rnd scale factor * C * (Y + D)
move    b0,y0             ; & reget left shift value
mpyr    b1,y1             ;mult by 2 again
asl     y:<cv,y0           ; & get C value

```

;ivquanti 3rd value:

```

mpy     y0,y1,a           a,x:(r1)+n1 ;3rd value: left justify bits

```

- 74 -

```

; & store 2nd data value
; move rsl to correct register
; Y + D
; forget sign extension
; C * (Y + D)
; & reget C factor

move    a0,a
add     x1,a
move    a1,y0
mpy     x0,y0,a      b0,y0

move    a,y1
mpyr    y0,y1,a      #>1,y1

asl     a          y:<bandcnt,b

move    a,x:(r1)+n1
jmp     <_getd_70

; All the 3 adjacent values in the sub-band are 0

_getd_60
clr     a          y:<bandcnt,b

move    #>1,y1
rep     #NPERGROUP
move    a,x:(r1)+n1

; We have just finished the current channel
; and if we just did the left, set up for the right channel
; if just did right channel, check for joint stereo and the
; intensity bound of sub-band
; if not a joint stereo frame, go set-up for the next sub-band.
; if right channel joint stereo sub-band intensity boundary reached,
; go set-up for the next sub-band.
; otherwise, decrement the intensity boundary sub-band counter
; before the go set-up for the next sub-band.

_getd_70
jclr    #LEFT_vs_RIGHT,y:<ctlflgs,_getd_72 ;if did left, go set-up right:
jclr    #JOINT_FRAMING,y:<ctlflgs,_getd_72 ;continue if not joint
jset    #JOINT_at_SB_BOUND,y:<ctlflgs,_getd_72 ;if reached, continue
sub     y1,b
move    b1,y:bandcnt
jgt     <_getd_72
bset    #JOINT_at_SB_BOUND,y:<ctlflgs ;if reached, set indicator

;after the left channel, set-up to do the right channel

_getd_72
move    #NUMSUBBANDS*NPERGROUP,n1
move    y:ivdata,r1
move    #>NUMSUBBANDS*NPERGROUP,a
move    y:<block,x0
add     x0,a      #NUMSUBBANDS,n3

bset    #LEFT_vs_RIGHT,y:<ctlflgs
move    (r1)+n1
move    a1,n2

;adj. to right channel fields
;get current start address
;move to SKFs for right channel
;get current block offset
;add right chan offset, set
;AND set adj to right SBindx
;indicate now doing right
;adjust r1 to right rec data
;offset register 2

; We have just finished both channels for a sub-band.
; 1. adjust left and right received sample pointers to next sub-band
; 2. increment SBindx array pointer for next sub-band
; 3. increment the SKFs array pointer over previous sub-band's 2nd & 3rd SKFs
; 4. increment the Allowed array pointer to next sub-band

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 75 -

```

_getd_75
    move    #>1,x0                ;incr left and right rev'd samps
    move    y:ivdata,a            ;address prev sub-band
    add     x0,a,(r3)+             ;adj next sub-band, incr SBindx
    move    a,y:ivdata            ;save new addr next sub-band
    move    #>16,x0               ;adj Allow ptr to next sub-band
    move    y:Allow,a             ;get current Allow address
    add     x0,a,#3,n2            ;adj Allow ptr, adj SKFs by 3
    move    a,y:Allow             ;save Allowed for next sub-band
    move    (r2)+n2               ;next sub-band SKFs addr

```

_getd_80

We have just finished a group of 3 samples per sub-band per channel
and we must send these value to the polysynthesis dsp

```

    move    r0,y:<svereg           ;save the key register
    bclr    #0,y:<not_appl        ;clear tested bit if not applic
    jsr     <synth                ;synth this group of values
    move    y:<svereg,r0          ;restore the key register

```

_getd_90

```

    bclr    #0,y:<not_appl        ;clear tested bit if not applic
    rts

```



- 76 -

opt fc.mex

; (c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.

; \DGCST\rsdecl6.asm: decoder Reed Solomon decoder

```

title 'RS Codec 64714 decoding program'
include 'box_ctl.asm'
include '..\Common\ioegu.asm'
include 'rctest.asm'

```

```

; this program will decode data in the input buffer according
; a decode profile with format as follow:

```

```

; parity byte, message byte, repetition times -- first block
; parity byte, message byte, repetition times -- 2nd block

```

```

; parity byte, message byte, repetition times, 0 -- last block

```

```

; the output data will be placed at output buffer

```

```

section highmisc
xdef pbyte
xdef mbyte
xdef cbyte
xdef dbyte
xdef inbyte
xdef mapbyte
xdef RsR3Tmp
xdef RsLpCnt
xdef RsLpCnt1

```

```

org yhe:
strdecl6_1_yhe

```

pbyte	ds	1	;parity byte
mbyte	ds	1	;message byte
cbyte	ds	1	;codeword byte
dbyte	ds	1	;delay byte
inbyte	ds	1	;insert zero byte
mapbyte	ds	1	;mess + pari byte
RsR3Tmp	ds	1	;tmp store r3
RsLpCnt	ds	1	;Rs Loop replacement
RsLpCnt1	ds	1	;Rs Loop replacement

```

endrdcl6_1_yhe
endsec

```

```

section highmisc
xdef PROF1
xdef CodeMinLen

```

```

; formula that cal the legacy delay
; (P)parity, (M)message, delay, repetition
; delay = (16*(P+M) + P*P + 4*P + 73) / 8 + 1

```

```

org yhe:

```

```

strdecl6_2_yhe
PROF1

```

```

dc 16,129,1

```

```

;RS profile
;RS decode

```

SUBSTITUTE SHEET (RULE 26)

BAA ORIGINAL



```

        dc      14,129,1          -77-
        dc      0,0,0
        dc      0,0,0,0

CodeMinLen
        dc      1,6,6,8,10,14,18,24,30,38,46      ;RS code min length per block
        dc      56,66,78,90,104,118              ;t=0,1,2,...,10
endrdcl6_2_yhe                                     ;t=11,12,...,16

        endsec

;*****
; RS decode routine
;*****

; This code is for RS decoder chip that the input is always enabled
; but output will be enabled when we have the output coming

; on entry
;   r1      :      output ptr in X SPACE
;   r3      :      input profile ptr in Y SPACE
;   r6      :      input data ptr in X SPACE

; on exit
;   r1      :      destroyed
;   r2      :      destroyed
;   r3      :      destroyed
;   r4      :      destroyed
;   r5      :      destroyed
;   r6      :      destroyed
;
;   a       :      destroyed
;   b       :      destroyed
;   x0      :      destroyed
;   x1      :      destroyed
;   y0      :      destroyed
;   y1      :      destroyed

        org     pli:

rsdcl6

;initial here

        move    #-1,m6              ;reset reg r6 to linear
        move    #0,n6              ;reset n6 to 0
        move    #-1,m1
        move    #3-1,m2              ;mod 3 -- 2,1,0
        move    #-1,m5
        move    #2,r2              ;set to first byte
        move    #0,r5              ;word count

        move    #>24,x0
        move    x0,y:rssc
        move    x:(r6)+,x0          ;set for rsgcrvalues
        move    x0,y:rscurwd

_Bentry
        bclr    #1,x:<<M_PCD        ;turn on the bit clk

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 78 -

```

        movep    #50808,x: <<M_BCR          ;set low to "cs" chip select
; SOFTWARE RESET
        clr      a                          ;zero
        move     a1,y:RSReg8                ;reset in case
; wait for some clock to pass away for the completeness of reset
        do       #400,_resetch
        nop
_resetch
; read message length and parity length from profile
        clr      a          y:(r3)+,x1      ;parity
        move     a,y:inbyte          ;set no insert byte
        move     x1,y:pbyte
        move     y:(r3)+,a1
        move     a1,y:mbyte          ;message length
;decide whether add zero is needed
        move     y:pbyte,a1            ;get parity byte
        lsr      a                    ;/2
        move     #CodeMinLen,r4        ;get min codelen
        move     a,n4                  ;get T
        move     y:mbyte,x1            ;get message byte len
        move     y:(r4+n4),a           ;get min len allowed
        cmp      x1,a
        jle      < NoInsert
        sub      x1,a
        move     a,y:inbyte            ;store insert byte num
_NoInsert
        move     y:inbyte,a            ;get inserted byte
        move     y:mbyte,x1
        add      x1,a          y:pbyte,x1 ;codewordlength=mbyte+pbyte+inbyte
        add      x1,a          ;codewordlength=mbyte+pbyte-inbyte
; wr RS block length
        move     a1,y:RSReg1
        move     a1,y:mapbyte
; a4=0,a3=1 only 40MHZ clk and CS and WR
; save message + parity byte
        move     y:mbyte,a            ;get message byte
        move     #>1,x1
        sub      x1,a          y:mbyte,x1 ;get message byte
        move     a1,y:cbyte            ;save message byte length -1
; cal the delay
        move     y:pbyte,x0            ;load x0
        mpy      x0,x0,a              ;p*p
        move     a0,a1
        lsr      a          #>73,x0    ;a == p**2
        add      x0,a          y:pbyte,b ;-- 73
        lsl      b          a1,x0
        lsl      b          y:mapbyte,a1
        add      x0,b              ;+ 4xp

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 79 -

```

        lsl      a          ;x 16
        lsl      a
        lsl      a
        lsl      a          b1,x0
        add      x0,a        #>1,x0      ;+ 16x(m+p)
        lsr      a          ;/8
        lsr      a
        lsr      a

; cal the delay

        sub      x1,a        y:pbyte,x1      ;get p byte
        sub      x1,a        y:inbyte,x1     ;get insert byte
        sub      x1,a
        move     a1,y:dbyte      ;delay without output reading
        move     y:pbyte,a1      ;# of bytes to be PARITY BYTES

; Wr parity length

        move     a1,y:RSReg2      ;a4=0,a3=1 clk CS/WR pulses are active
        lsr      a              ;/2 get correction power

; Wr correction power, t number

        move     a1,y:RSReg3      ;a4=0,a3=1 only reset pulse and clk
        move     #>32,a1          ;set SYMBOL Synthesis of the RS codec

; Wr synthesis clock

        move     a1,y:RSReg6      ;N at address 5
        move     #>0,a1          ;set SYMBOL division 8 bit per symbol

; Wr bit per symbol

        move     a1,y:RSReg7      ;address 6

; reset again after all register have been filled

        move     #0,a1
        move     a1,y:RSReg8      ; reset again

; wait for some time

        do       #400,_resetch2
        nop
_resetch2                                ;40 MHZ clk is there

        bset     #1,x:<<M_PCD      ;turn off the bit clk after reset

; Initialization is completed

        movep     #S0101,x:<<M_BCR      ;set low duration of "cs"(chip select)

; RS decoding start

        move     y:(r3)+,x0        ;load the repetition time
        move     x0,y:RsLpCnt
        move     r3,y:RsR3Tmp      ;save r3 for later

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 80 -

```

_RsLoop
; get first input byte

    move    #8,n4
    jsr     <rsgetvalues

; or FRAME START SIGNAL and first byte

    move    #>$100,x1          ;insert frame start signal
    or      x1,a              ;The first DATA byte is "OR" gated
                                ;as the R-S codec thinks you are
                                ;sending the first data byte at
                                ;the same time with the FRAME
                                ;start pulse.

    do      #8,_dtasnd100
    movep    a1,y:<<RSIN        ;SEND 1st data byte and also RAISE the
                                ; FRAME START PULSE
_dtasnd100

; input message-1 byte to decode

    clr     a                  y:cbyte,x0
    move     x0,y:RsLpCnt1      ;initial loop count

_RsLoop1
    move     #8,n4
    jsr     <rsgetvalues
    do      #8,_dtasnd1
    movep    a1,y:<<RSIN        ;a4=1,a3=1 only clk and data
_dtasnd1
    move     y:RsLpCnt1,a        ;test loop cnt
    move     #>1,x0              ;dec count
    sub      x0,a
    jle     <_EndRsLoop1
    move     a,y:RsLpCnt1        ;resave loop count
    jmp     <_RsLoop1

_EndRsLoop1

; insert zero message byte to decode if it's not zero

    move     y:inbyte,a          ;chk if insertion is needed
    tst      a
    jeq     <_NoIntion

    clr     a                  y:inbyte,x0
    move     x0,y:RsLpCnt1      ;initial loop count

_RsLoop2
    do      #8,_dtasnd3
    movep    a1,y:<<RSIN        ;a4=1,a3=1 only clk and data
_dtasnd3
    move     y:RsLpCnt1,a        ;test loop cnt
    move     #>1,x0              ;dec count
    sub      x0,a
    jle     <_EndRsLoop2
    move     a,y:RsLpCnt1        ;resave loop count
    clr     a
    jmp     <_RsLoop2

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 81 -

```

_EndRsLoop2
_NoIntion
; input parity byte to decode
    clr      a      y:pbyte,x0
    move     x0,y:RsLpCnt1      ;initial loop count
_RsLoop3
    move     #8,n4
    jsr      <rsgetvalues
    do       #8,_dtasnd5
    movep    a1,y:<<RSIN      ;a4=1,a3=1 only clk and data
    _dtasnd5
    move     y:RsLpCnt1,a      ;test loop cnt
    move     #>1,x0      ;dec count
    sub     x0,a
    jle     <_EndRsLoop3
    move     a,y:RsLpCnt1      ;resave loop count
    jmp     <_RsLoop3
_EndRsLoop3
; push zero input for delay byte
    clr      a      y:dbyte,x1
    move     x1,y:RsLpCnt1      ;initial loop count
_RsLoop4
    do       #8,_Gdata100
    movep    a1,y:<<RSIN      ;a4=1,a3=1 only clk and data
    _Gdata100
    move     y:RsLpCnt1,a      ;test loop cnt
    move     #>1,x0      ;dec count
    sub     x0,a
    jle     <_EndRsLoop4
    move     a,y:RsLpCnt1      ;resave loop count
    clr     a
    jmp     <_RsLoop4
_EndRsLoop4
; reading decoded data output
    move     y:mbyte,x1
    move     #>$80,y0      ;shift right 16 bits
    move     #>$8000,y1      ;shift right 8 bits
    move     x1,y:RsLpCnt1      ;initial lp count
_RsLoop5
    clr      a      #>$ff,x0
    do       #8,_Gdata200
    movep    a1,y:<<RSIN      ;a4=1,a3=1 only clk and data
    _Gdata200
    move     y:RSOUT,b1      ;provide clock and read data
    and     x0,b
    move     b1,x0      ;get set for shift
; test byte counter and put output byte to right pos of output buffer

```



- 82 -

```

        move    r2,a                ;get byte count
        move    #>2,x1
        cmp     x1,a    #>1,x1
        jne     <_Tndbyte

; fst byte

        mpy     x0,y1,a #>Sff0000,x0    ;shift right 8 bits
        clr     b
        move    a0,b1
        and     x0,b
        move    b1,x:(r1)
        jmp     <_EndABYTE

_Tndbyte
        cmp     x1,a    #0,x1
        jne     <_Lstbyte

        mpy     x0,y0,a #>Sff00,x0    ;shift right 16 bits
        clr     b
        move    a0,b1
        and     x0,b    x:(r1),x1
        or      x1,b    ;or it with previous 8 bits
        move    b1,x:(r1)
        jmp     <_EndABYTE

_Lstbyte
        clr     b
        move    #>Sff,b1    ;mask off last 8 bits
        and     x0,b    x:(r1),x1
        or      x1,b    (r5)+    ;increase word count
        move    b1,x:(r1)+    ;save the musicam data for desort

_EndABYTE
        move    (r2)-    ;2-1-0 mod
        move    y:RsLpCnt1,a    ;test loop cnt
        move    #>1,x0    ;dec count
        sub     x0,a
        jle     <_EndRsLoop5
        move    a,y:RsLpCnt1    ;resave loop count
        jmp     <_RsLoop5
_EndRsLoop5

; forget inserted zero message byte next

        move    y:inbyte,a    ;chk if insertion is needed
        tst     a
        jeq     <_NoIntion10

        clr     a
        move    y:inbyte,x0
        move    x0,y:RsLpCnt1    ;initial lp count

_RsLoop6
        do      #8,_dtasnd20
        movep    a1,y:<<RSIN    ;a4=1,a3=1 only clk and data
        _dtasnd20
        move    y:RsLpCnt1,a    ;test loop cnt
        move    #>1,x0    ;dec count
        sub     x0,a
        jle     <_EndRsLoop6

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 83 -

```

        move    a,y:RsLpCnt1      ;resave loop count
        clr     a
        jmp     <_RsLoop6
_EndRsLoop6

_NoIntion10
; forget parity output at the end of frame

        clr     a      y:pbyte,x1
        move    x1,y:RsLpCnt1      ;initial lp count

_RsLoop7
do      #8,_Gdata300
movep   a1,y:<<RSIN      ;a4=1,a3=1 only clk and data
_Gdata300
move    y:RsLpCnt1,a      ;test loop cnt
move    #>1,x0      ;dec count
sub     x0,a
jle     <_EndRsLoop7
move    a,y:RsLpCnt1      ;resave loop count
clr     a
jmp     <_RsLoop7
_EndRsLoop7

        move    y:RsLpCnt,a      ;test loop cnt
        move    #>1,x1      ;dec count
        sub     x1,a
        jle     <_RepEnd
        move    a,y:RsLpCnt      ;resave loop count
        jmp     <_RsLoop

; repetition end

_RepEnd
        move    y:RsR3Tmp,r3      ;reload profile ptr
        nop
        move    y:(r3),a      ;test if a '0' at last RS block
        tst     a
        jne     <_Bentry

; patch zero to make 96 (a full frame)

        move    #>96,a
        move    r5,x0
        sub     x0,a      #0,x0
        jle     <_PatchZero1
do      a,_PatchZero1
        move    x0,x:(r1)+      ;inc to next frame
_PatchZero1

; end of RS decoding for One Profile

        move    #-1,m2
        movep   #S0001,x:<<M_BCR      ;set all external io wait states

        rts

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 84 -

```

opt    f=

; (c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \DGCST\bitalloc.asm: use the o_psych parameter (safety margin)
; This routine is used to allocate the bits.
; It allocates at least some bits to all sub-bands with a positive SMR.
; It allocates in three phases:
;   A. allocate all sub-bands until they are all below
;       the Global Masking Threshold (regardless as to how many
;       bits it takes)
;       note 1. a limit (sub-band boundary) is set which requires
;               all sub-bands up to the boundary require at least
;               index 1 be allocated even if the signal is already
;               below the Global Masking Threshold. (This provides
;               a noticeable improvement in continuity of sound)
;       After Phase A is completed, a test is made to see if the bit pool
;       was overflowed by the allocation.
;       a. if the frame fits, Phase B is skipped and Phase C is done
;       b. otherwise, Phase B is required to selectively de-allocate the
;          best sub-band candidates.

; on entry
;   y:<stereo = flags:
;   (set on entry) bit 0 indicates whether or not left channel active
;                   0 = channel not active
;                   1 = channel active for framing
;   bit 1 indicates whether or not center channel active
;                   0 = channel not active
;                   1 = channel active for framing
;   bit 2 indicates whether or not right channel active
;                   0 = channel not active
;                   1 = channel active for framing
;   bit 3 is used to indicate left vs right channel
;       applies if bit 4 set to 0 (NOT center channel)
;                   0 = looping through left channel arrays
;                   1 = looping through right channel arrays
;   bit 4 is used to indicate center channel vs left right
;                   0 = process left or right channel arrays
;                   1 = looping through center channel arrays
;   bit 5 is used as the FirstTime switch in an allocation
;                   0 = cleared if any allocations were made
;                   1 = no allocations made to any sub-bands
;   bit 6 is used for critical de-allocate and allocate passes:
;       with below masking threshold being a criteria
;       de-allocate:
;                   0 = select from any sub-band channel
;                   1 = select from only those below mask
;       allocate:
;                   0 = there are sub-band channels not below mask
;                   1 = all sub-bands are below mask
;   bit 7 is used for critical de-allocate and allocate passes:
;       de-allocate:
;                   0 = select from any sub-band channel
;                   1 = select from those with 2 or more allocation
;       allocate:
;                   0 = are sub-bands not below hearing thresh
;                   1 = all sub-bands are below hearing thresh
;   bit 8 is used for critical de-allocate and allocate passes:

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGIN.

- 85 -

```

;
; de-allocate:
;     0 = select from any sub-band channel
;     1 = select from any sub-band channel
; allocate: for final pass after bit allocation timer
;     0 = timer interrupt not yet sensed
;     1 = timer interrupt was sensed
; bit 9 is to simply indicate that the sub-band limit for
; allocating at least ONE position has been reached
; within a current loop:
;     0 = NOT at sub-band limit
;     1 = reached the sub-band limit
; bit 10 is to simply indicate that the maximum sub-band for
; consideration for allocation has been reached
; within a current loop:
;     0 = NOT at maximum sub-band limit
;     1 = reached the maximum sub-band limit
;
; y:audbits = number of bits available for sbits, scale factors and data
; y:usedsb = number of sub-bands actually used
; y:<limitsb = number of sub-bands requiring at least one allocation
; y:<qttalloc = timer interrupt set to signal quit allocation loops
; r0 = addr of the SBits array (x memory)
; r1 = addr of MinMasking Db array (x memory)
; r2 = addr of SubBandMax array (x memory)
; r4 = addr of the SubBandPosition array (x memory)
; r5 = addr of the SubBandIndex array (x memory)
;
; on exit
; a = destroyed
; b = destroyed
; x0 = destroyed
; x1 = destroyed
; y0 = destroyed
; y1 = destroyed
; r3 = destroyed
; r6 = destroyed
; n0 = destroyed
; n1 = destroyed
; n2 = destroyed
; n3 = destroyed
; n4 = destroyed
; n5 = destroyed
; n6 = destroyed
;
; AtLimit array by sub-bands (32):
; bit 0 set when allocation is below the masking threshold
; bit 1 set when allocation is below the threshold of hearing
; bit 2 set when allocation is at the limit of maximum position
; or there are not enough bits to allocate
; the sub-band further
;
; include 'def.asm'
; include 'box_ctl.asm'
;
; section lowmisc
;
; xdef MNRsub
; xdef AvlBits
; xdef TotBits
; xdef HldBits

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 86 -

```

        xdef      count

        org      yli:
stbitalloc_yli

MNRsub ds      1          ;count of entries in de-allocate tables
AvlBits ds     1          ;available bits to allocate
TotBits ds     1          ;current bit count allocated
HldBits ds     1          ;sub-band critical allocation
count ds      1          ;sub-band counter

endbitalloc_yli
endsec

        section highmisc

        xdef      BitsAdd
        xdef      BPosAdd
        xdef      BinxAdd
        xdef      AllwAdd
        xdef      MaxPos
        xdef      MNRsb
        xdef      MNRmin
        xdef      MNRinx
        xdef      MNRpos

        org      yhe:
stbitalloc_yhe

BitsAdd ds      1          ;save address of SBits array
BPosAdd ds      1          ;save address of SBPosition array
BinxAdd ds      1          ;save address of SBIndex array
AllwAdd ds      1          ;save addr of applicable Allowed table
MaxPos ds      1          ;Max Position per selected Allowed table
MNRsb ds      1          ;curr sub-band for allocation
MNRmin ds      1          ;value of curr sub-band for allocation
MNRinx ds      1          ;new index for selected sub-band
MNRpos ds      1          ;new allowed position for selected sb

endbitalloc_yhe
endsec

        section highmisc

        xdef      AtLimit
        xdef      SBMSr
        xdef      SBMNRmax
        xdef      MNRval
        xdef      MNRsbc

        org      xhe:
stbitalloc_xhe

;flags set when a sub-band reaches its limit of allocation:
; (one per 32 subbands)
;   bit 0: set if below the global masking threshold
;   bit 1: set if not used or fully allocated

AtLimit ds      NUMSUBBANDS

```

- 87 -

;This array holds the MinMaskingDb - SubBandMax for each of the 32 subbands

SBMSr ds NUMSUBBANDS ;Mask-Signal ratio by sub-band

;This array holds the deallocation selection values:

; (MinMaskingDb - SubBandMax) + SNR[position at next lower index]
;for each of the 0-31 subbands

SBMNRmax ds NUMSUBBANDS ;Mask-to-Signal ratio
; plus SNR[PrevPos]

MNRval ds NUMSUBBANDS ;table of ordered values sub-band

MNRsbc ds NUMSUBBANDS ;table of associated sub-band

endbitalloc_xhe
endsec

section xtables

xdef ndatabit

xdef NDataBit

xdef NSKFBits

xdef SNR

org xhe:

stbitalloc_xtbl

;This is the addr of the selected table, ISO or CCS compression,
; for the number of bits for data allocation by position

ndatabit ds 1 ;addr ISO or CCS compress NDataBit tbl

;This is the ISO table for the number of bits for data allocation by position

NDataBit

dc	0*NUMPERSUBBAND	;index = 0, no transmit	= 0 bits
dc	5*NUMPERSUBBAND	;index = 1, packed	= 60 bits
dc	7*NUMPERSUBBAND	;index = 2, packed	= 84 bits
dc	9*NUMPERSUBBAND	;index = 3	= 108 bits
dc	10*NUMPERSUBBAND	;index = 4, packed	= 120 bits
dc	12*NUMPERSUBBAND	;index = 5	= 144 bits
dc	15*NUMPERSUBBAND	;index = 6	= 180 bits
dc	18*NUMPERSUBBAND	;index = 7	= 216 bits
dc	21*NUMPERSUBBAND	;index = 8	= 252 bits
dc	24*NUMPERSUBBAND	;index = 9	= 288 bits
dc	27*NUMPERSUBBAND	;index = 10	= 324 bits
dc	30*NUMPERSUBBAND	;index = 11	= 360 bits
dc	33*NUMPERSUBBAND	;index = 12	= 396 bits
dc	36*NUMPERSUBBAND	;index = 13	= 432 bits
dc	39*NUMPERSUBBAND	;index = 14	= 468 bits
dc	42*NUMPERSUBBAND	;index = 15	= 504 bits
dc	45*NUMPERSUBBAND	;index = 16	= 540 bits
dc	48*NUMPERSUBBAND	;index = 17	= 576 bits

;This is the CCS compression table for number of bits
; for data allocation by position

dc	0*NUMPERSUBBAND	;index = 0, no transmit	= 0 bits
dc	4*NUMPERSUBBAND	;index = 1, packed	= 48 bits
dc	6*NUMPERSUBBAND	;index = 2, packed	= 72 bits

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 88 -

dc	8*NUMPERSUBBAND	;index = 3	= 96 bits
dc	10*NUMPERSUBBAND	;index = 4, packed	= 120 bits
dc	12*NUMPERSUBBAND	;index = 5	= 144 bits
dc	15*NUMPERSUBBAND	;index = 6	= 180 bits
dc	18*NUMPERSUBBAND	;index = 7	= 216 bits
dc	21*NUMPERSUBBAND	;index = 8	= 252 bits
dc	24*NUMPERSUBBAND	;index = 9	= 288 bits
dc	27*NUMPERSUBBAND	;index = 10	= 324 bits
dc	30*NUMPERSUBBAND	;index = 11	= 360 bits
dc	33*NUMPERSUBBAND	;index = 12	= 396 bits
dc	36*NUMPERSUBBAND	;index = 13	= 432 bits
dc	39*NUMPERSUBBAND	;index = 14	= 468 bits
dc	42*NUMPERSUBBAND	;index = 15	= 504 bits
dc	45*NUMPERSUBBAND	;index = 16	= 540 bits
dc	48*NUMPERSUBBAND	;index = 17	= 576 bits

;Each sub-band, if it is transmitted, must send scale factors. The
;Sbit patterns determine how many different scale factors are transmitted.
;The number of scale factors transmitted may be 0, 1, 2 or 3. Each scale
;factor requires 6 bits.

;Sbit patterns

00	Transmit all three scale factors	18 (3 * 6 bits)
01	Transmit the second two scale factors	12 (2 * 6 bits)
10	Transmit only one scale factor	6 (1 * 6 bits)
11	Transmit the first two scale factors	12 (2 * 6 bits)

;The NBits array is used to determine the number of bits to allocate for the
;scale factors. NSBITS (the 2 bits for SBITS code) are added to account for
;all required scale factor bits (18+2,12+2,6+2,12+2).

NSKFBits

dc 20,14,8,14

;This is the table for Signal to Noise ratio by position

include '..\xmicro\snr.asm'

endbitalloc_xtbl
endsec

org phe:

bitalloc

;Save the array starting addresses

move	r0,y:BitsAdd	;save register of SBits array
move	r4,y:BPosAdd	;save register of SubBandPosition array
move	r5,y:BINxAdd	;save register of SubBandindex array

;select the ISO or CCS compression table for NDataBit:

move	#NDataBit,r5	;standard ISO table
move	#18,n5	;offset to CCS compression table
jclr	#0,y:<cmprscctl,_bita_20_A	;if not applicable, continue
move	(r5)+n5	;select the CCS compression table

_bita_20_A

move r5,x:ndatabit ;set addr of NDataBit table for alloc

- 89 -

```

;set up the MNR array
    move    #SBMSr,r5                ;addr of Mask-to-Signal by sub-band

;apply the safety factor
    move    y:o_psych,y0            ;get the safety factor

;loop through all sub-bands
    do      #NUMSUBBANDS,_bita_30_A
    move    x:(r2)+,x0                ;get a channel SBMax
    move    x:(r1)+,b                ;get its channel MinMsk
    sub     x0,b                    ;MinMask = SBMax - Mask-to-Signal ratio
    sub     y0,b                    ;apply safety factor to channel value
    move    b,x:(r5)+                ;store for test if below mask already

_bita_30_A                            ;END of do loop

;set the working value for bits available for allocation
    move    y:audbits,x0            ;get standard available bit cnt
    move    x0,y:<AvlBits            ;store as working bit cnt

_bita_40_A

; (c) TotBits = 0;                    /* start the bit allocation counter */
;
    clr     a                        #>1,x1    ;total bit used, x1 = 1 for start index
    move    a,y1                    ;y1 = 0 to initialize
    move    a,y:<TotBits
    move    a,y:<count                ;start the sub-band counter
    bclr    #AT_LIMIT_SUBBAND,y:<stereo    ;NOT yet at sub-band limit
    ; which require at least 1 allocation
    bclr    #AT_USED_SUBBAND,y:<stereo    ;NOT yet at sub-band maximum
    ; limit for coding used sub-bands

;initial allocation for all sub-bands;
; 1. that are within the use (less than UsedSubBands)
; 2. with a MinimumMasking to MaximumSignal above the masking threshold
;
    move    #SBMNRmax,r0            ;addr of de-alloc Max signal-noise
    move    #SBMSr,r1                ;addr of Mask-to-Signal by sub-band
    move    y:BitsAdd,r2            ;set register of SBits array
    move    y:AllwAdd,n3            ;init the current Allow table
    move    y:BPosAdd,r4            ;set register of SubBandPosition array
    move    y:BInxAdd,r5            ;set register of SubBandIndex array
    move    #AtLimit,r6            ;point to SubBandAtLimit array

;clear the n registers for the channel reference
    clr     a                        #0,n0
    move    a,n1                    ;SBMNRmax array
    move    a,n2                    ;SBMSr array
    move    a,n3                    ;SBits array
    move    a,n4                    ;SBPos array
    move    a,n5                    ;SBInx array
    move    a,n6                    ;AtLimit array

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 90 -

```

;initial allocation pass
;do all required sub-bands
    do      #NUMSUBBANDS,_bita_990_A
;initialize the pertinent sub-band values to 0
        move    y1,x:(r6+n6)      ;clear allocated limit flag 'AtLimit'
        move    y1,x:(r5+n5)      ;clear allocated index 'SBindx'
        move    y1,x:(r4+n4)      ;clear allocated position 'SBPos'
;if we reached the used sub-band limit,
; take this one out of the picture completely
        jset     #AT_USED_SUBBAND,y:<stereo,_bita_180_A
        move     y:<count,y0      ;get current sub-band (00-31)
;see if we reached the used sub-band limit
        move     y:<usedsb,b      ;get count of used subbands for testing
        cmp      y0,b            ;see if sub-band not to be coded
        jgt      <_bita_50_A      ;if not, continue
        bset     #AT_USED_SUBBAND,y:<stereo ;just reached sub-band maximum
        jmp      <_bita_180_A      ;take completely out of use
_bita_50_A
;if we reached the sub-band limit for those requiring at least one sub-band.
; see if we have anything to allocate to get below the Global Masking Threshold
        jset     #AT_LIMIT_SUBBAND,y:<stereo,_bita_90_A
;see if at least one allocation is required regardless of signal to noise ratio
        move     y:<limitsb,a      ;get sub-band limit for at least 1 alloc
        cmp      y0,a            ;if there is initial allocation
        jgt      <_bita_95_A      ;continue
        bset     #AT_LIMIT_SUBBAND,y:<stereo ;just reached that limit
_bita_90_A
;otherwise, see if below Mask-to-Signal
        move     x:(r1+n1),a      ;get sub-band's Mask-to-Signal ratio
        tst      a              ;test Mast-to-Sig for positive value
        jgt      <_bita_190_A      ;if below masking thresh, set flag
_bita_95_A
;find Signal-to-Noise position that puts Signal below Masking Threshold
        move     x1,r7            ;start at 1st Signal-to-Noise position
        move     #SNR,n7          ;addr of Signal-to-Noise table
        move     x:(r1+n1),y0      ;get signal to mask ratio
        do      #NUMSNRPOSITIONS-1,_bita_110_A
        move     x:(r7+n7),a      ;get the Signal-Noise at position
        add      y0,a            ;add MNR to SNR for test

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 91 -

```

        jle      <_bita_100_A      ;still above mask, try next position
;now below the Global Mask, quit the loop

        enddo      ;found position, stop #NUMSNRPOS-1 loop
        jmp      <_bita_110_A      ;go to end of loop

_bita_100_A
; try the next position and continue the loop

        move      (r7)+      ;try next Sig-Noise position

_bita_110_A
;END of #NUMSNRPOSITIONS-1 dc loop

        move      r7,y0      ;save the matched SNR position
        move      y:MaxPos,a  ;to test if exceeded max position
        cmp      y0,a      y1,r3 ;is counted position greater than max
                                ; & start at index 0 with allocation
        jge      <_bita_115_A    ;if not, go on to match the index
        move      a1,y0      ;set position at the maximum position

_bita_115_A
;find index of the position that best matches the selected SNR position

        do      #NUMINDEXES,_bita_130_A

        move      x:(r3+n3),a    ;get the sub-band indexed position
        cmp      y0,a      ;compare to selected position
        jlt      <_bita_120_A    ;match not found yet, try next index

;found the matching index, quit the loop

        enddo      ;found index, stop #NUMINDEXES loop
        jmp      <_bita_130_A    ;go to end of loop

_bita_120_A
;try the next index and continue the loop

        move      (r3)+      ;try position at next index

;see if end of the table line reached

        move      x:(r3+n3),a    ;get this next index to test
        tst      a      ;test for an index of zero
        jne      <_bita_125_A    ;if not 0, keep looking

;index of zero indicates no higher indices apply, back up 1 and use that

        move      (r3)-      ;use previous index
        bset      #ALLOCATE_LIMIT,x:(r6+n6) ;set the completely allocated bit
        bset      #HEARING_LIMIT,x:(r6+n6) ;set the completely allocated bit
        move      x:(r3+n3),a    ;assign the last index position
        enddo      ;found index, stop #NUMINDEXES loop
        jmp      <_bita_130_A    ;go to end of loop

_bita_125_A
        nop      ;keep looping

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 92 -

```

_bita_130_A                                ;END of #NUMINDEXES do loop
;set the initial allocation SubBandIndex and SubBandPosition
        move    r3,x:(r5+n5)                ;set initial allocation SBIndx
        move    a1,x:(r4+n4)                ;set initial allocation SBPos
;determine the number of scale factor bits allocated at this position
        move    x:(r2+n2),n7                ;get the SBits scale factor code (0-3)
        move    #NSKFBits,r7                ;addr SBits scale factor bit count tbl
        nop
        move    x:(r7+n7),y0                ;save the scale factor bit count
_bita_140_A
;add the bits required for the signal data
        move    x:(r4+n4),n7                ;get the position
        move    x:ndatabit,r7                ;address of data bit count by position
        nop
        move    x:(r7+n7),a                  ;get the bit count at this position
        add     y0,a      y:<TotBits,x0      ;add scale factor bits
        ; and get curr TotBits
        add     x0,a                          ;update TotBits with bits just allocated
        move    a,y:<TotBits                  ;save new allocated total bits
;check that Signal-to-Noise position that Signal below Masking Threshold
        move    #SNR,r7                      ;addr of Signal-to-Noise table
        move    x:(r1+n1),y0                ;get signal to mask ratio
        move    x:(r7+n7),a                  ;get the Signal-Noise at position
        add     y0,a      x:(r5+n5),r3      ;add MNR to SNR for test
        ; & set up to set prev index for its pos
        jle     <_bita_160_A                 ;above mask, skip next statement
        bset    #MASKING_LIMIT,x:(r6+n6)    ;set AtLimit partially done allocate
_bita_160_A
;set the value for testing the best sub-band to deallocate bits from
;if the frame cannot handle the full required allocation
        move    (r3)-                        ;back up one index to get that position
        move    x:(r3+n3),n7                ;get the position at the previous index
        nop
        move    x:(r7+n7),a                  ;get the Signal-Noise at position
        add     y0,a                          ;calc Sig-to-Noise at prev position
        move    a,x:(r0+n0)                  ;save in SBMNRmax array for later
        jmp     <_bita_200_A                 ;continue with the next sub-band
_bita_180_A
;sub-band is not to be coded at all
        bset    #ALLOCATE_LIMIT,x:(r6+n6)    ;set AtLimit totally out of allocation
        bset    #HEARING_LIMIT,x:(r6+n6)    ;set AtLimit at threshold of hearing
_bita_190_A

```

- 93 -

;sub-band is set to indicate it is at its masking threshold

bset #MASKING_LIMIT,x:(r6-n6) ;set AtLimit partially done allocate

_bita_200_A

;finished the sub-band set up for the initial allocation of the next subband

```

move (r0)+ ;next sub-band SBMNRmax
move (r1)+ ;next sub-band SBMSr
move #16,r3 ;to position to next Allowed sb table
move (r2)+ ;next sub-band SBbits
move (r3)+n3 ;next sub-band Allowed table array
move r3,n3 ;set addr for next sub-band Allowed pos
move (r4)+ ;next sub-band SBPos
move (r5)+ ;next sub-band SBindx
move y:<count,r7 ;get current sub-band count
move (r6)+ ;next sub-band AtLimit
move (r7)+ ;increment the sub-band counter
move r7,y:<count ;save new sub-band

```

_bita_990_A

;END of #NUMSUBBANDS do loop

; done with the initial allocation phase, phase A

; set the de-allocation passes initial state of control flags

```

bset #MASKING_PASS,y:<stereo ;flag do masking passes
bclr #HEARING_PASS,y:<stereo ;allocate index must be > 1
bclr #FINAL_PASS,y:<stereo ;NOT final passes

```

;see if frame fits or do we have to de-allocate selectively

```

move y:<TotBits,x0 ;get the total bits allocated
move y:<AvlBits,a ;get available bits
cmp x0,a ;TotBits vs BitsAvailable
jge <_bita_990_B ;it fits, allocate any leftover bits
dc #1000,_bita_990_B

```

```

;test the bit allocation timeout flag
; if the timer flag was trip, switch over to the final bit allocation
; of any remaining bits

```

```

clr #0,y:<qtalloc,_bita_10_B
set #FINAL_PASS,y:<stereo,_bita_10_B ;continue, if final
bset #FINAL_PASS,y:<stereo ;set for FINAL criteria
enddc ;stop the #1000 loop and exit
move y:<TotBits,x0 ;get the total bits allocated
jmp <_bita_990_C ;out of time, de-alloc under last basis

```

_bita_10_B

;now let's look for qualifying candidates for next de-allocation

```

move #SBMNRmax,r0 ;addr of de-alloc Max signal-noise
move y:BinxAdd,r5 ;set register of SubBandIndex array
move #AtLimit,r6 ;point to SubBandAtLimit array
move #0,n0 ;offset to the channel SBMNRmax
move n0,n5 ;offset to chan SBindx

```


- 94 -

```

move    n0,n6                ;offset to chan AtLimit
move    #0,r2                ;use r2 as a sub-band counter
move    r2,y:<MNRsub          ;start cnt of de-allocate table entries
move    #>1,x1                ;to test for index of 1
move    y:<limitsb,y1         ;to test for at least one alloc limit
move    #MNRval,n3           ;get address of MNRval table
move    #MNRsbc,n4           ;get address of MNRsbc table

;to deallocate the 1 index if the signal starts out below global mask

move    #SBMSr,r1             ;addr of Mask-to-Signal by sub-band
move    n0,n1                ;offset to chan SBMSr

;loop thru the sub-bands
do      y:<usedsb,_bita_60_B

;if no index has been allocated, try the next sub-band

move    x:(r5+n5),a           ;check for an allocated index
tst     a                    ;if zero, try the next sub-band
jeq     <_bita_70_B          ;no allocation try next sub-band

;if the 3rd mode of selection, no checks are made
jset    #FINAL_PASS,y:<stereo,_bita_60_B    ;3rd mode, use this one

;if 2nd mode of selection sub-band may be below the masking threshold, but
; checks to make sure that if index allocated is ONE and that the
; sub-band is not required for continuity
jset    #HEARING_PASS,y:<stereo,_bita_50_B    ;2nd mode num of index

;must be 1st mode of selection which requires that the sub-band
; be below the masking threshold
jle     #MASKING_LIMIT,x:(r6+n6),_bita_70_B    ;skip: above mask thresh

_bita_50_B

;if we have allocated only 1 index, skip this sub-band if at least one
; allocation is required

cmp     x1,a                  ;see if index at 1
jgt     <_bita_60_B           ;no, this sub-band qualifies

move    r2,a                  ;get current sub-band
cmp     y1,a                  ;see if sub-band below at least 1
jge     <_bita_70_B           ;if greater, deallocation candidate
move    #>14,y1               ;if greater than 14, check
cmp     y1,a                  ;test sb vs 14, restore limitsb to y1
jlt     <_bita_70_B           ;if less than 14, keep the 1 allocation
move    x:(r1+r1),b           ;get Max Signal to MinMask
tst     b                    ;if positive, started below global mask
jle     <_bita_70_B           ;if not positive, keep the 1 allocation

_bita_60_B

;candidate qualifies,
; insert this candidate into the table for initial de-allocation

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 95 -

```

        jsr      <insert_value
_bita_70_B
;advance to the next sub-band

        move     (r2)+          ;increment the sub-band counter
        move     (r0)+          ;next sub-band SBMNRmax
        move     (r5)+          ;next sub-band SBindx
        move     (r6)+          ;next sub-band AtLimit

_bita_80_B                                ;end of y:<usedsb do loop
;if there are any entries in the de-allocate tables, start reclaiming bits

        move     y:<MNRsub,a      ;get the de-allocate table entry cnt
        tst      a              ;test for zero, no entries
        jne      <_bita_110_B    ;are entries at this criteria, dealloc

;since there were no candidates to deallocate (MNRsub = 0),
;change the selection criteria:
;   if we've done the final criteria and nothing to de-allocate,
;   we can do nothing here, exit (How Come???)
;   if we've not found anything with at least 2 indexes allocated,
;   switch to select from any sub-bands
;   if we've not found anything below the masking threshold,
;   switch to at least 2 indexes alloc
;redo the selection criteria

        jset     #FINAL_PASS,y:<stereo,_bita_095_B ;??? shouldn't be, exit
        jset     #HEARING_PASS,y:<stereo,_bita_100_B
        jset     #MASKING_PASS,y:<stereo,_bita_105_B
        bset     #MASKING_PASS,y:<stereo
        jmp      <_bita_200_B    ;loop thru with this criteria

_bita_095_B
        enddo                ;stop the #1000 loop and exit
        move     y:<TotBits,x0  ;get the total bits allocated
        jmp      <_bita_990_C

_bita_100_B
        bclr     #HEARING_PASS,y:<stereo
        bset     #FINAL_PASS,y:<stereo
        jmp      <_bita_200_B    ;loop thru with this criteria

_bita_105_B
        bclr     #MASKING_PASS,y:<stereo
        bset     #HEARING_PASS,y:<stereo
        jmp      <_bita_200_B    ;loop thru with this criteria

;there are entries in the de-allocate tables
_bita_110_B
;de-allocate from the table from 1st entry to last
; or until enough bits have been reclaimed

        clr      a
        move     a,y:<count      ;start counter thru the table

```



- 96 -

;loop through the ordered de-allocation table

```

do      y:<MNRsub,_bita_19C_B
move    #MNRsbc,n0      ;address of MNRsbc table
move    y:<count,r0      ;current table entry index
nop
move    x:(r0+n0),a      ;get selected sub-band
move    a,y:MNRsb        ;store current sub-band (0-31)
move    (r0)+            ;increment to next table entry
move    r0,y:<count      ;save next table entry

```

;restore the channel array addresses

```

move    #SBMNRmax,r0      ;addr of de-alloc Max signal-noise
move    #SBMSr,r1          ;addr of Mask-to-Signal by sub-band
move    y:BitsAdd,r2       ;set register of SBits array
move    y:BPosAdd,r4        ;set register of SubBandPosition array
move    y:BinxAdd,r5        ;set register of SubBandIndex array
move    #AtLimit,r6        ;point to SubBandAtLimit array

```

;set the proper allowed table of indexed position based on the selected sub-band

```

move    y:AllwAdd,r3      ;init the current Allow table
tst     a                  ;see if it's sub-band zero (from above)
jeq     <_bita_150_B       ;sub-band zero was selected
move    #16,n3             ;to increment to next sub-band addr
do      a,_bita_150_B      ;increment to sub-band number chosen
move    (r3)+n3            ;16 position entries per sub-band

```

_bita_150_B

```

move    r3,n3              ;set Allowed addr for sub-band chosen
move    y:MNRsb,n0         ;get selected sub-band in SBMNRmax
move    n0,n1              ;sub-band in SBMSr
move    n0,n2              ;sub-band in SBits
move    n0,n4              ;sub-band in SBPos
move    n0,n5              ;sub-band in SBIndx
move    n0,n6              ;sub-band in AtLimit
move    x:ndatabit,r7      ;address of data bit count by position
move    y:<TotBits,a        ;get current bits allocated
move    x:(r5+n5),r3        ;get the current allocated index
move    x:(r4+n4),n7        ;get the position at the old index
move    (r3)-              ;back up one index
move    r3,x:(r5+n5)        ;save new SBIndx for sub-band
move    x:(r7+n7),x0        ;data bits allocated at that position
sub     x0,a               ;subtract old allocated data bits
move    x:(r3+n3),n7        ;get new position
move    n7,x:(r4+n4)        ;save new SBPos for sub-band
move    x:(r7+n7),b        ;data bits allocated at new position
add     b,a                ;add new allocated data bits

tst     b                  ;see if index 1 just de-allocated
jne     <_bita_160_B       ;if not, save the new TotBits value

```

;we have to take off the scale factor bits

```

move    x:(r2-n2),n7        ;get the SBits scale factor code (0-3)
move    #NSKFBits,r7        ;addr SBits scale factor bit count tbl
nop

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 97 -

```

        move    x:(r7+n7),y0      ;get the scale factor bit count
        sub     y0,a              ;subtract from TotBits

_bita_160_B
        move    a,y:<TotBits      ;save the new total bits

;check if Signal-to-Noise position that Signal above/below Masking Threshold

        bclr    #MASKING_LIMIT,x:(r6+n6) ;clear AtLimit below masking threshold
        move    x:(r4+n4),n7      ;get the position
        move    #SNR,r7          ;addr of Signal-to-Noise table
        move    x:(r1+n1),y0      ;get signal to mask ratio
        move    x:(r7+n7),a       ;get the Signal-Noise at position
        add     y0,a      x:(r5+n5),r3 ;add MNR to SNR for test
                                   ; & set up to set prev index for its pos
        jle     <_bita_170_B      ;above mask, skip next statement
        bset    #MASKING_LIMIT,x:(r6+n6) ;set AtLimit below masking threshold

_bita_170_B
;check if the bit pool can now handle the frame as allocated

        move    y:<TotBits,a      ;get the new total bits
        move    y:<AvlBits,x0     ;get the available bits
        cmp     x0,a              ;BitsAvailable vs TotBits
        jgt     <_bita_180_B      ;need more, continue with de-allocation

        enddo                    ;we're done here, stop MNRsub loop
        enddo                    ;we're done here, stop #1000 loop
        jmp     <_bita_990_B

_bita_180_B
;if there is no index allocated (r3 = 0), continue with the next table entry

        move    r3,a              ;get newly decremented index allocated
        tst     a      (r3)-      ;if it is zero, continue
                                   ; & back up one index for that position
        jeq     <_bita_185_B      ;allocated index equals 0, continue

;set the value for testing the best sub-band to deallocate bits from
;if the frame cannot handle the full required allocation

        move    x:(r3+n3),n7      ;get the position at the previous index
        nop
        move    x:(r7+n7),a       ;get the Signal-Noise at position
        add     y0,a              ;calc Sig-to-Noise at prev position
        move    a,x:(r0+n0)       ;save in SBMNRmax array for later

_bita_185_B
        nop                      ;continue y:MNRsub do loop

_bita_190_B
        nop                      ;end of y:MNRsub do loop

_bita_200_B
        nop                      ;continue #1000 do loop

_bita_990_B
        nop                      ;end of #1000 do loop

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 98 -

```

; set the allocation passes initial state of control flags
      bset    #MASKING_PASS,y:<stereo      ;flag do masking passes
      bclr    #HEARING_PASS,y:<stereo      ;NOT hearing threshold passes
      bclr    #FINAL_PASS,y:<stereo        ;NOT final passes

; get the total bits allocated so far
      move    y:<TotBits,x0

; Now that we have the initial bit allocation, iterate on it
; (c) for( LoopCount = 0; ; ++LoopCount ) {
      do      #1000,_bita_990_C

; test the bit allocation timeout flag
; if the timer flag was trip, switch over to the final bit allocation
; of any remaining bits
      jclr    #0,y:<qtalloc,_bita_10_C
      jset    #FINAL_PASS,y:<stereo,_bita_10_C
      bset    #FINAL_PASS,y:<stereo

; this is equivalent to the call to the c subroutine:
; (c) AllocateBits()
; initial allocation is done, set-up for as needed allocation loop
; restore the left channel array addresses
_bita_10_C
      move    #SBMSr,r1      ;set register of SBMSr array
      move    y:BitsAdd,r2    ;set register of SBits array
      move    y:BPosAdd,r4    ;set register of SubBandPosition array
      move    y:BinxAdd,r5    ;set register of SubBandIndex array
      move    #AtLimit,r6    ;point to SubBandAtLimit array

; (c)      FirstTime = 1;      /*start run thru subbands this time */
      bset    #FIRST_TIME,y:<stereo      ;FirstTime = 10

; clear the n registers for the channel reference
      clr     a
      move    a1,y:<count      ;start the sub-band counter
      move    y:AliwAdd,rC
      move    #SNR,r3
      move    a,n1            ;SBMSr array
      move    a,n2            ;SBits array
      move    a,n4            ;SBPos array
      move    a,n5            ;SBindx array
      move    a,n6            ;AtLimit array

; go through all used sub-bands looking at only those
; that have not reached the allocation limit
      dc      y:<usedsb,_bita_130_C

```

BAD ORIGINAL

SUBSTITUTE SHEET (RULE 26)

- 99 -

```

;see if this sub-band's limit flag was set previously, and skip if it has
jset    #ALLOCATE_LIMIT,x:(r6+n6),_bita_100_C ;skip subband reached limit
jset    #FINAL_PASS,y:<stereo,_bita_40_C ;pass skips below mask check
jset    #MASKING_LIMIT,x:(r6+n6),_bita_100_C ;skip subband reached limit

_bita_40_C
move    x:(r4+n4),a ;get curr position[SubBand]

;see if this sub-band has reached its limit already
move    y:MaxPos,y0 ;set max value
cmp     y0,a,a1,n3 ;see if max position: move pos to n3
jeq     <_bita_80_C ;reached its allocation limit, set flag

;check this sub-band out
;see if there is room to handle the next allocation for this sub-band
clr     b #>1,y1 ;init added scale factor bits
; & to incr to next allowed bits size
move    x:(r5+n5),a ;SubBandIndex[SubBand]

;if this will be the 1st index, we must account for the scale factor bits
tst     a #NSKFBits,r7 ;see if 0
; & set addr of NSKFBits array
jne     <_bita_50_C ;not 1st index, skip add scale bits

;set the scale factor + sbits needed for this 1st index in this sub-band
move    x:(r2+n2),n7 ;get SBits index
nop
move    x:(r7+n7),b ;num bits for scaling info

_bita_50_C
add     y1,a,x:ndatabits,r7 ;incr, get addr of NDataBits
move    a1,n0 ;set offset for Allowed next index

;see if next allocation is passed the max for this sub-band as per Allowed table
nop
move    x:(r0+n0),a ;get the NextPosition as the new pos
tst     a,a1,n7 ;see if passed the maximum position
; & move new pos to n7
jeq     <_bita_80_C ;reached its allocation limit, set flag

;test the allocation at this new position
move    x:(r7+n7),y1 ;get NDataBits[NextSBPos]
add     y1,b,n3,n7 ;add to any scaling info bits
; & set offset SubBandPos[SubBand];
move    b1,y1 ;bits to add for next index
move    x0,b ;b==>TestBits = OldTotBits
move    x:(r7+n7),y0 ;get NDataBits[SBPos[SubBand]]
sub     y0,b,a1,x1 ;TestBits -= current bits
; & put new position in proper reg

```



-100-

```

add     y1,c      y:<AvlBits,a      ; TestBits -- next allocation bits
                                ; & gets BitsAvailable

;(c)      if( TestBits > BitsAvailable ) {
;(c)          AtLimit = 1;
;(c)          continue;
;(c)      }

cmp     b,a      b,y:TotBits      ;see if room & save allocation
jlt     <_bita_80_C      ; no room, set as AtLimit and continue

; if this is the final loop, skip the next test and allocate the bits

jset    #FINAL_PASS,y:<stereo,_bita_70_C ;pass skips below mask check

;(c)      SMR = SubBandMax[SubBand]
;(c)          - MinMaskingDb[SubBand]
;(c)      MNR = SNR[SubBandPosition[SubBand]] - SMR

move     x:(r3+n3),y1      ;get SNR[SubBandPos[SubBand]]
move     x:(r1-n1),a      ;SBMSr[SubBand] Mask-to-Signal
add     y1,a      y:MNRmin,b      ;add Sig-Noise ratio;
                                ; & get MNRmin for below
jgt     <_bita_90_C      ;below Masking, go to take out partially

move     a,y1      ;save MNR
jset    #FIRST_TIME,y:<stereo,_bita_60_C ;if first, save as minimum
cmp     y1,b      ;MNRmin - MNR
jle     <_bita_100_C

_bita_60_C
move     n0,y:MNRinx      ;MNRinx = NewIndex;
move     x1,y:MNRpos      ;MNRpos = NewPosition;
move     y:<TotBits,x1      ;get the allocation of bits
move     x1,y:<HldBits      ;save the allocation of bits
move     y:<count,x1      ;get current sub-band
move     x1,y:MNRsb      ;MNRsb = SubBand;
move     y1,y:MNRmin      ;MNRmin = MNR;
bclr     #FIRST_TIME,y:<stereo ;clear FirstTime flag
jmp     <_bita_100_C

; we are on the final allocations passes after all sub-bands
; are driven below the Global Masking threshold

_bita_70_C
move     y:<TotBits,x0      ;save new TotBits
move     n0,x:(r5+n5)      ;save new sub-band index
move     x1,x:(r4+n4)      ;save new allocation position
bclr     #FIRST_TIME,y:<stereo ;clear FirstTime flag
jmp     <_bita_100_C

_bita_80_C
bset     #ALLOCATE_LIMIT,x:(r6+n6) ;set the completely allocated bit
bset     #HEARING_LIMIT,x:(r6+n6) ;set the completely allocated bit

_bita_90_C
bset     #MASKING_LIMIT,x:(r6+n6) ;set the reached global masking bit

_bita_100_C

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 101 -

```

        move    y:<count,r7          ;get current sub-band to increment
        move    #16,n0              ;now update Allowed to next sub_band
        move    (r1)+                ;SBMSr array
        move    (r2)+                ;SBits array
        move    (r4)+                ;SBPos array
        move    (r5)+                ;SBIndx array
        move    (r6)+                ;AtLimit array
        move    (r0)+n0              ;advance Allowed to next sub-band
        move    (r7)+                ;increment the sub-band counter
        move    r7,y:<count          ;save new sub-band number

_bita_130_C
; At this point the following registers are in use
; y:AvlBits = # of bits available
; y:MNRsb = MNRsb
; y:MNRMin = MNRmin

; We test now to see if this trip thru the loop produced any changes
; and if not, we have finished the bit allocation for this frame.
; (c) if ( FirstTime )
; (c) return;

jclr    #FIRST_TIME,y:<stereo,_bita_140_C ;not 1st. alloc to selected
jclr    #FINAL_PASS,y:<stereo,_bita_160_C ;not final, set 1 more loop

;finished, end the loop and go to exit routine

enddo
jmp     <_bita_990_C

_bita_140_C
;test flag all candidates are below masking threshold

jset    #FINAL_PASS,y:<stereo,_bita_170_C ;if final, allocated already

;restore the channel array addresses

move    y:BPosAdd,r4                ;set register of SubBandPosition array
move    y:BInxAdd,r5                ;set register of SubBandIndex array

;
SubBandIndex[MNRsb]++
SubBandPosition[MNRsb] = AllowedPositions[MNRsb][SubBandIndex[MNRsb]]

move    y:MNRsb,n5                  ;MNRsb
move    n5,n4                       ;MNRsb
move    y:MNRinx,x1                  ;get the saved new index
move    x1,x:(r5+n5)                 ;update the SBIndx for selected sub-band
move    y:MNRpos,x1                  ;get the saved new Allowed position
move    x1,x:(r4+n4)                 ;update the SBPos for selected sub-band
move    y:<HldBits,x0                ;set the new bit allocation total cnt
jmp     <_bita_170_C                ;continue major loop

;now lets just allocate what's left now that all are below mask

_bita_160_C
bset    #FINAL_PASS,y:<stereo      ; just loop now

```

BAD ORIGINAL

- 102 -

_bita_170_C
nop

_bita_990_C

```

move    x0,y:<TotBits      ;save bits actually allocated
move    y:<AvlBits,b       ;determine number of bits padded
sub     x0,b               ;bits available minus total allocated
move    b1,y:padbits      ;save count of unallocated audio bits

```

rts

;insert_value():

```

;This routine orders the table of values per sub-band
;that are to be de-allocated as needed. The table is ordered in
;descending sequence that makes the 1st entry the one that can best
;afford a deallocation.

```

;on entry:

```

x:(r0+n0) = the current value to be inserted
r2 = the sub-band number to be inserted
y:MNRsub = current count of entries in the ordered deallocation tables
n3 = address of MNRval table
n4 = address of MNRsbc table

```

;on exit:

```

y:MNRsub = incremented count of entries in ordered deallocation tables

```

```

a = destroyed
b = destroyed
x0 = destroyed
y0 = destroyed
r3 = destroyed
r4 = destroyed

```

org phe:

insert_value

```

;get the current value to be inserted and set up the start into
;the ordered table of values and the associated table of sub-band

```

```

move    x:(r0+n0),a       ;get the current value to insert
move    y:<MNRsub,b       ;get current count of table entries

```

```

;if this is the 1st value to be inserted into the table, skip the
;search for its place and enter this as table entry no 1

```

```

tst     b      #0,r3      ;see if this is 1st entry into table
                        ; & set to 1st entry in MNRval table
jeq     <_insert_50      ;if 1st, skip following table search

```

```

;search through the table of entries so far established looking for where
;to store this current value

```

```

dc      y:<MNRsub,_insert_20

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 103 -

```

move    x:(r3)+n3,x0      ;get the table value for comparison
cmp     x0,a              ;against the new value to be inserted
jlt     <_insert_10       ;if less, value is further down table

;when the new value is greater than or equal to the table entry,
;this is its place in the table, we may have to shift the following
;table entries in order to enter this new value

        enddo             ;stop the y:MNRsub do loop
        jmp      <_insert_20 ;see if the table must be shifted

_insert_10
        move     (r3)-      ;try the next table entry

_insert_20
        ;end of y:MNRsub do loop

;if this entry number (its place in the table) equals the count of entries,
;this entry will be the new LAST entry in the table

        move     r3,x0      ;get its place in the table to compare
        cmp     x0,b        ;its place to current table entry count
        jgt     <_insert_25 ;if less, we have to shift the table
        jeq     <_insert_50 ;if eq, entry is appended to the table
        move     b1,r3      ;?? let's make sure we use last entry
        jmp     <_insert_50

_insert_25

;we need to shift the subsequent entries in the table down one and then
;insert this new sub-band value

        move     b1,r3      ;establish the curr table ends
        move     b1,r4      ;for both MNRval and MNRsbc
        move     (r3)+n3     ;set r3 with addr of MNRval end + 1
        move     (r4)+n4     ;set r4 with addr of MNRsbc end + 1
        move     (r3)-      ;back off 1 to get last MNRval entry
        sub      x0,b       ;number of table entries to shift
                           ; & back off 1 to get last MNRsbc entry

        do       b,_insert_40 ;shift each down 1 position in tables

        move     x:(r3)+,y0   ;get curr value and incr to rec addr
        move     y0,x:(r3)-   ;put value 1 entry down & back up 1
        move     x:(r4)+,y0   ;curr sub-band/chan & incr to rec addr
        move     y0,x:(r4)-   ;put value 1 entry down & back up 1
        move     (r3)-        ;back up one more entry table MNRval
        move     (r4)-        ;back up one more entry table MNRsbc

_insert_40
        ; end of b do loop

;restore entry location to receive value and sub-band

        move     x0,r3

_insert_50

;insert the current value at this location in the ordered table
;also insert the sub-band number

        move     r3,r4      ;matching position in the MNRsbc table

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 104 -

```
move    a,x:(r3+n3)          ;enter sorted value
move    r2,x:(r4+n4)          ;enter the sub-band number
;increment the count of entries in the ordered deallocation tables

move    y:<MNRsub,r3          ;we need to increment entry counter
nop
move    (r3)+
move    r3,y:<MNRsub          ;save the new table entry count
rts
```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 105 -

```

opt      fc
; (c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \DGCST\botsallo.asm

title 'Initialize bit output'

; This routine is used to initialize the bit output routines.

include 'def.asm'
include 'box_ctl.asm'

section lowmisc
xdef    sc,curwd

org     yli
stbitsallo_yli

sc      ds      1           ;shift count
curwd   ds      1           ;current word

endbitsallo_yli
endsec

org     phe:

;bitpool()
; This subroutine determines the number of bits available based
; on the output bit rate and the type of framing
;
;.....
;The table below is based on a Sampling Rate at 48,000 /sec and shows
;the breakdown of bit counts based on bit rate o/p and choice of frame type
;
;.....
;kb      frame      Mono      Full      Stereo      4-bound      8-bound      12-bound      16-bound
;rate    bits      fix avail  fix avail  fix avail  fix avail  fix avail  fix avail  fix avail
;-----
;384     9216      136   9080   224   8992   152   9064   168   9048   183   9033   195   9021
;256     6144      .    6008   .    5920   .    5992   .    5976   .    5961   .    5949
;192     4608      .    4472   .    4384   .    4456   .    4440   .    4425   .    4413
;128     3072      .    2936   .    2848   .    2920   .    2904   .    2889   .    2877
;112     2688      .    2552   .    2464   .    2536   .    2520   .    2505   .    2493
;96      2304      .    2168   .    2080   .    2152   .    2136   .    2121   .    2109
;64      1536      .    1400   .    1312   .    1384   .    1368   .    1353   .    1341
;56      1344      136   1208   224   1120   152   1192   168   1176   183   1161   195   1149
;.....

; y: <stereo = flags:
; test bit indicating applicablaiton of CRC-16 protection
; 0 = NOT APPLICABLE
; 1 = CRC-16 protection APPLIES

; y: frmbits = the total number of bits in a frame at the specified
; bit rate

; on exit:
; x0 destroyed = returned number of required (fixed) bits
; x1 destroyed = returned number of bits available for bit allocation

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

```

; a destroyed
; r0 destroyed
; r1 destroyed
; r3 destroyed

; - 106 -

org phe:

bitpool

; Select the proper Allowed table:
; ISO:
; 1. for low sampling rates (24 or 16 K):
;     set ISO Extension Allowed table (Allowed_3)
; 2. for high sampling rates (48, 44.1 or 32 K):
;     a. based on MAXSUBBANDS less than 27,
;         set ISO lower bit rate Allowed table (Allowed_2)
;     b. else,
;         set ISO higher bit rate Allowed table (Allowed_1)
; CCS:
;     set ISO higher bit rate Allowed table (Allowed_1)

; low sampling rate:
; test the frame header ID bit (if 0, it's a low sampling rate frame)

move    #smplidbit,r0      ;addr of frame header ID bit (0 = low)
nop
jset    #0,y:(r0),_bitp_000_A ;if high rate, select Allowed table

move    #Allowed_3,r0      ;addr of low sampling allowed table
move    #skftbl_3,r1       ;addr of the BAL bits table
move    #>15,x1            ;maximum position Allowed_3 table
jmp     <_bitp_010_A       ;go to store Allowed table address

_bitp_000_A

; high sampling rate:
; set the proper Allowed table address based on working MAXSUBBANDS (y:<maxubs)
; if less than 27, used table 2

move    y:<maxsubs,x0      ;get current MAXSUBBANDS
move    #>27,a            ;to see which of 2 tables applies
move    #>17,x1           ;maximum position Allowed_1 table
move    #skftbl_1,r1      ;addr of the BAL bits table
cmp     x0,a #Allowed_1,r0 ;see if need the low bit rate table
; & set up as Allowed_1 table
jle     <_bitp_010_A      ;Allowed_1 table applies

; select the lower bit rate Allowed table

move    #Allowed_2,r0      ;addr of the BAL bits table
move    #skftbl_2,r1       ;maximum position Allowed_2 table
move    #>16,x1

_bitp_010_A

; set the address of the selected Allowed table
; set the address of the selected BAL's bit table
; set the maximum position code

```



- 107 -

```

move    r1,y:AllwAdd
move    r1,x:skftbl
move    x1,y:MaxPos

;determine the bits required for ancillary data (taken from audio pit pool):
; start with bits required to store the padded data byte count in frame

move    #>BITSFORPADDING,b      ;bits in the padded byte count
move    y:maxbytes,y1           ;get max bytes at baud rate
move    y:<bytecnt,a            ;get current count of bytes received
cmp     y1,a    #>BITSPERBYTE,x1 ;see max versus current count
; & set multiplier
jge     <_bitp_00               ;if more than max, can only send max
move    a,y1                    ;less than max, send all received

_bitp_00

;multiply the bytecount for bits per byte

mpy     x1,y1,a                 ;to get the required bit
asr     a    y1,y:<bytesfrm      ;shift integer result
; & set byte count for framing

move    a0,a
add     a,b                     ;add to the count of bytes

move    b,y:ancbits             ;set ancillary data bit count

;set the number of fixed bits used, and the number of available bits for audio

clr     a    #0,x1              ;0 a as accum, zero CRC checksum bit cnt

;set the address and bit offsets to identify the end of the current full frame
; and set the end of the formatted frame

move    y:<frmnext,r1           ;address for start the next frame
move    y:<outsize,m1           ;circular ctrl addr the framing o/p buf

;set the fixed bits for the audio frame

move    #>NSYNC,x0              ;number of SYNC bits
add     x0,a    #>NSYST,x0      ;plus number of bits in frame system hdr
add     x0,a    x:skftbl,r0     ;get base of used bits table

jclr    #PROTECT,y:<stereo,_bitp_35 ;skip checksum bits if no protect
move    #>NCRCBITS,x1           ;add applicable bits for the checksum

_bitp_35
add     x1,a                    ;add checksum protection, if any

;account for the bits required for protection encoding

move    #>REED_SOLOMON_BITS,x1  ;bits required for Kadir's routine
add     x1,a                    ;add protection bits to fixed bit cnt

;accumulate the bit allocation bits for standard number of sub-bands
; included in the frame for the left and right (if applicable)

do      y:<maxsubs,_bitp_50

;accumulate for the channel

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 108 -

```

        move    x:(r0)+,x1
        add     x1,a

_bitp_50
        move    a,x0                ;return fixed bits
        move    y:frmbits,b        ;total size of frame in bits

;subtract any bits required for ancillary data

        move    y:anchbits,y1
        sub     y1,b

_bitp_80
        sub     a,b                ;total bits - fixed bits
        move    b,x1              ;return number of audio data bits avail

;now determine word and bit offsets for the end of the audio frame

        add     y1,b                ;restore bits for ancillary data
        add     a,b    #>24,y1    ;restore to full audio frame size
                                   ; & set number bits in a word
        move    y:<frmstrt,r1      ;count words to last word in frame

_bitp_90
        cmp     y1,b                ;see if reached last word
        jlt     <_bitp_100          ;if so, set eoframe word & bit offsets
        sub     y1,b    (r1)+
        jmp     <_bitp_90

_bitp_100
        move    r1,y:audendw        ;to identify end of audio part of frame
        move    b,y:audendb        ;bit offset end of audio part of frame
        move    y:<linear,m1        ;reset to linear buffer control

        rts

;bitsalloc
; This subroutine starts the bit allocation of values into the
; frame buffer values are inserted by setvalue() and by bitfree() below

; on exit
; y:<sc = 0
; y:<curwd = initialized (0) 1st word in frame buffer
; a = destroyed

bitsalloc
        move    #0,a
        move    a,y:<sc              ;initialize the shift count
        move    a,y:<curwd          ;initialize curwd (1st bit in op frame)

        rts

        page

;bitfree
; This routine flushes the last bits to the output buffer

; on entry
; r6 = address of next word the output frame buffer is memory

```



- 109 -

```

; on exit:
;   a = destroyed
;   b = destroyed
;   x0 = destroyed
;   x1 = destroyed
;   y0 = destroyed
;   y1 = destroyed

section highmisc
xdef audendw
xdef audendb

org yhe:
sbitsallc_yhe

audendw ds 1 ;address of end of audio portion of frame
audendb ds 1 ;bit offset to end of audio portion of frame

endbitsallc_yhe
endsec

bitsfree

;see if all of the frame has been output totally

move y:<frmnxt,x1 ;get address for start of next frame
move r6,b ;next o/p address of current frame
cmp x1,b #>24,a ;if addresses = start, done
; and set up for the next test
;frame done, exit
jeq <_free_90

;see if the last word of the frame is to be output next

move y:<frmlast,x1 ;last word address of current frame
cmp x1,b y:<sc,x0 ;test if address = last word
; and get number of bits in last word
;last word, chk block seq number needed
jeq <_free_20

;output last partially formatted data word before zero fill remainder of frame

sub x0,a #>24,x0 ;get number of bits left
cmp x0,a #0,x0 ;24 - number of bits left
jeq <_free_05 ;not partially formatted y:sc == 0

move y:<curwd,b ;get current output word
rep a ;output the necessary # of bits
lsl b

move b1,x:(r6) ;save in the output
move x0,y:<sc ;zero the current bit offset

_free_05
clr a ;output zero for remainder of frame

_free_10

;see if the last word of the frame is to be output next

move r6,b ;next o/p address of current frame

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 110 -

```

      cmp      x1,b
      jeq      <_free_20
      move     a1,x:(16)
      jmp      <_free_10

_free_20
      move     #0,y0
      move     #0,x0
      move     #>24,a
      move     y:<sc,y1
      sub      y1,a
      sub      x0,a
      tst      a
      jle      <_free_90

      move     a,n4
      jsr      <setvalue

      _free_90
      rts

```

;see if last word next
 ;last word, chk block seq number needed
 ;output frame word and increment addr
 ;continue to flush the buffer

 ;init with zeros to pad last word
 ;init with no bits req for seq number
 ;bits in the word
 ;get current formatted word offset
 ;bits remaining
 ;bits required for block seq num
 ;test if any zero bits to output
 ;if none, try the block seq num

 ;number of bits to output
 ;pad word with zeroes as needed



- 111 -

```

opt      fc,mex

(c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.
\DGCS\xmircmus.asm: Reed Solomon version for DigiCast

title    'Micro MUSICAM Transmitter Main'

; (7/23/92) xmicro.asm micro MONO version of XPSYCHO and XCODE combined

include 'def.asm'
include '..\common\ioequ.asm'
include 'box_ctl.asm'

section lowmisc

xdef     word_out
xdef     word_in

xdef     startyli
xdef     not_appl
xdef     maxsubs
xdef     oldccs
xdef     usedsb
xdef     stereo
xdef     cmprscctl
xdef     oprptr
xdef     outmus
xdef     outsize
xdef     frmstrt
xdef     frmnext
xdef     frmlast
xdef     timer
xdef     timeout
xdef     qtalloc
xdef     ipwptr
xdef     polyst
xdef     nmskfregs
xdef     maxcritbnds
xdef     linear
xdef     junk
xdef     endyli

xdef     dbgcnt
xdef     limitsb

org      yli:

stxmicro_yli

word_out ds      1      ;applicable hardware output (leds, switches)
word_in  ds      1      ;applicable hardware input (switches, lines)

startyli

not_appl ds      1      ;satisfy non-applicable hardware settings

maxsubs ds      1      ;working MAXSUBBRANDS for sample/bit rate
oldccs  ds      1      ;encode MPEG-ISO or old CCS CDQ1000's
                        0 = MPEG-ISO

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 112 -

```

usedsb ds 1
stereo ds 1

; 1 = old CCS CDQ1000's
; number of used sub-bands
; y: < stereo = flags:
; bit 0 means stereo vs mono framing
; 0 = stereo framing
; 1 = mono framing
; bit 1 indicates left vs right channel
; 0 = looping thru left channel arrays
; 1 = looping thru right channel arrays
; bit 2 indicates joint stereo applies
; 0 = NOT joint stereo framing type
; 1 = IS joint stereo framing type
; bit 3 indicates curr frame upgraded to
; full stereo by joint bit allocation
; (if joint stereo applies)
; 0 = normal joint stereo allocation
; 1 = FULL STEREO allocation
; bit 4 indicates the stereo intensity
; sub-band boundary has been reached
; (if joint stereo applies)
; 0 = NO sub-bands still below
; intensity boundary
; 1 = sub-bands above intensity
; boundary
; bit 5 is FirstTime switch in a loop
; thru the bit allocation
; 0 = cleared if any allocations
; were made
; 1 = no allocations made to any
; sub-band
; bit 6 indicates a below masking
; threshold allocation pass
; 0 = some sub-bands not below mask
; 1 = all sub-bands are below mask
; bit 7 indicates a below hearing
; threshold allocation pass
; 0 = some sub-bands not below hearing
; threshold
; 1 = all sub-bands are below hearing
; threshold
; bit 8 indicates final bit allocation
; passes to use up any available bits
; 0 = not yet
; 1 = allocate remainder in bit pool
; bit 9 indicates limit of sub-bands requiring
; at least one position has been reached:
; 0 = not yet, 1 = limit reached
; bit 10 indicates maximum limit of sub-bands
; that are to be allocated has been reached:
; 0 = not yet, 1 = limit reached

cmprsc1 ds 1
; control flag for CCS compression:
; bit 0 = application:
; 0 = ISO standard
; 1 = CCS compression applies

oprptr ds 1
outmus ds 1
outsize ds 1
frmstrt ds 1
frmnext ds 1
; read pointer into output frame buffer
; number of words to read in
; circular buffer ctl frame o/p buffer
; starting addr of current frame
; starting addr of next frame

```

SUBSTITUTE SHEET (RULE 26)

BAB ORIGINAL

- 113 -

```

frmlast ds      1      ;last word addr of current frame
timer ds        1      ;0.024/0.036 msec timer interrupt sensor
timeout ds      1      ;0.024/0.036 msec timer interrupt exception
qtalloc ds      1      ;0.024/0.036 msec timer interrupt bit alloc
                    ; signal bit allocator to finish up
ipwptr ds       1      ;write pointer into input inpcm buffer
polyst ds       1      ;addr of the polyanalysis start
nmskfregs ds     1      ;NMSKFREQS based on selected sample rate
maxcritbnds ds   1      ;MAXCRITBND5 based on selected sample rate
linear ds       1      ;reset mX as linear buffer control

junk ds         1      ;!!!debug

endyli

dbgcnt dc       0      ;!!!debug counter of flag
limitsb dc      0      ;LIMITSUBBANDS ;sub-bands req at least 1 allocation

endxmicro_yli

endsec

section ptable

xdef ptable
xdef a_psych,b_psych
xdef c_psych,d_psych
xdef e_psych,f_psych,g_psych
xdef h_psych,i_psych,j_psych
xdef k_psych,l_psych,m_psych,n_psych,o_psych,p_psych
xdef q_psych,r_psych,s_psych,t_psych,u_psych,v_psych,w_psych,x_psych
xdef y_psych,z_psych
xdef z1_psych,z2_psych,z3_psych,z4_psych,z5_psych,z6_psych

org yli:
stptable_yli

ptable
;this table is known as IRT

a_psych dc 0.0467146 ;A curval= 9 dB
b_psych dc 0.0498289 ;B curval= 3 dB/Bark
c_psych dc 0.0259526 ;C curval= 5 dB
d_psych dc 0.0498289 ;D curval= 3 dB/Bark
e_psych dc 0.0882387 ;E curval= 17 dB/Bark
f_psych dc 0.4000000 ;F curval= 4 1/Bark
g_psych dc 0.0311431 ;G curval= 6 dB/Bark
h_psych dc 0.0882387 ;H curval= 17 dB/Bark
i_psych dc 0.0882387 ;I curval= 17 dB/Bark
j_psych dc 0.1000000 ;J curval= 1 1/Bark
k_psych dc 0.0000000 ;K curval= 0.0000000
l_psych dc 0.0000000 ;L curval= 0.0000000
m_psych dc 0.0000000 ;M curval= 0.0000000
n_psych dc 0.0000000 ;N: CCS compression = NO < .5 >= YES
o_psych dc 0.0000000 ;O curval= 0.0000000
p_psych dc 0.0000000 ;P curval= 0.0000000
q_psych dc 0.0000000 ;Q curval= 0.0000000
r_psych dc 0.0000000 ;R curval= 0.0000000
s_psych dc 0.0000000 ;S curval= 0.0000000

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 114 -

t_psych	dc	0.0000000	;T curval=	0.0000000
u_psych	dc	0.0000000	;U curval=	0.0000000
v_psych	dc	0.0000000	;V curval=	0.0000000
w_psych	dc	0.0000000	;W curval=	0.0000000
x_psych	dc	0.0103810	;X curval=	2 dB/Bark
y_psych	dc	0.0259525	;Y curval=	5 dB/Bark
z_psych	dc	0.0415239	;Z curval=	8 dB/Bark
z1_psych	dc	0.0000000	;Z1 curval=	0.0000000
z2_psych	dc	0.0000000	;Z2 curval=	0.0000000
z3_psych	dc	0.0000000	;Z3: 4 to 30 =	used sub-bands (mono)
z4_psych	dc	0.0000000	;Z4 curval=	0.0000000
z5_psych	dc	0.0000000	;Z5 curval=	0.0000000
z6_psych	dc	0.0000000	;Z6 curval=	0.0000000

endtable_yli
endsec

section highmisc

xdef startyhe

xdef bitrate
xdef frmrate
xdef smplcde
xdef smplrte
xdef smplidbit
xdef bndwdth
xdef frmtype
xdef opfrtyp
xdef baudrte
xdef oputcde
xdef frmbits
xdef fixbits
xdef audbits
xdef anchbits
xdef stintns
xdef b_i
xdef fmap
xdef ThresSLB
xdef Threshld
xdef cb
xdef g_cb
xdef dbaddtbl
xdef plctmn

xdef endyhe

xdef sampling
xdef bitrates
xdef baudclk

org yhe:

stxmico_yhe

startyhe

bitrate ds 1

;bit rate code for MUSICAM frame header
; sampling rate 48 K or 32 K:
; ISO and old CCS CDQ1000:

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 115 -

			3 (0011) = 56 KBits
			4 (0100) = 64 KBits
			sampling rate 24 K or 16 K:
			ISO:
			7 (0111) = 56 KBits
			8 (1000) = 64 KBits
			old CCS CDQ1000:
			3 (0011) = 56 KBits
			4 (0100) = 64 KBits
frmrate ds	1		overall frame bit rate as to hardware
			switches (1 bit) indicate
			bit rate sets numb words in a frame:
			0 = low Kbit rate
			1 = high Kbit rate
smplcde ds	1		sample rate code in MUSICAM header:
			ISO:
			00 = 44.1 K or 22.05 K
			01 = 48 K or 24 K
			10 = 32 K or 16 K
			old CCS CDQ1000:
			00 = 16 K
			01 = 48 K
			10 = 32 K
			11 = 24 K
smplrte ds	1		PCM data sampling rate: low vs high rate
			depending on flag in box ctrl.asm that
			indicates the pairing (16/24, 16/32, 16/48,
			24/32, 24/48 or 32/48)
			switches (1 bit) indicate
			0 = 16000, 24000 or 32000
			1 = 24000, 32000 or 48000
smplidbit	ds	1	hdr id bit:
			ISO:
			1 for 44.1, 48, and 32 K sample rates
			0 for 22.05, 24, and 16 K sample rates
			old CCS CDQ1000:
			1 is always used with special sample
			rate codes in the header (above)
bndwdth ds	1		code for setting sub-band limits
frmtype ds	1		dip switches (2 bits) are set to:
			11 = (3) mono (1 channel)
opfrtyp ds	1		current frame type after bit allocation
baudrte ds	1		ancillary data baud rate
cpucode ds	1		type of output coding: MUSICAM vs G722
			switches (1 bit) indicate
			0 = MUSICAM frames
			1 = G722 data
frmbits ds	1		bits in the audio portion of frame
fixbits ds	1		bits required before audio data bits
audbits ds	1		number of bits available for audio data
ancbits ds	1		bits required for ancillary data current frame
stntns ds	1		intensity subband boundary code
b_i	ds	1	addr b_i table for low or high sample rate
fmap	ds	1	addr fmap table for low or high sample rate
ThresSLB	ds	1	addr ThresSLB table for low or high sample rate
Threshld	ds	1	addr Threshld table for low or high sample rate
cb	ds	1	addr cb table for low or high sample rate
g_cb	ds	1	addr g_cb table for low or high sample rate
dbAddtbl	ds	1	addr DBAddtbl
picmn ds	:		successive phase lock detect high center main



- 116 -

```

endyhe
;table of sampling rates
    SAMPLERATES
;table of bit rates
    BITRATES
;baud rate table for ancillary data
    BAUDCLK
endxmicro_yhe
    endsec
    org    phe
start
; The external wait state is set to 1. This allows the HCT541's to
; put their data on the bus in plenty of time.
    movep    #0001,x:<<M_BCR    ;set all external io wait states
;set dsp56002 clock to selected MHz (PLL Control Register)
    XCODE_M_PCTL
; PORT C Assignments
;
; s = ssi port
; i = input port
; o = output port
;
    XCODE_PORT_C_M_PCC    ;set port C control register
    XCODE_PORT_C_M_PCD    ;set output data to port C
    XCODE_PORT_C_M_PCDDR    ;set port C data direction reg
; initialize the ssi port for the ad converter
    XCODE_SSI_M_CRA    ;set ssi cra register
    XCODE_SSI_M_CRB    ;set ssi crb register
; initialize the sci port for tty
    XCODE_SCI_M_SCR    ;set sci status control register
; PORT B Assignments
;
; 14 13 12 - 11 10 9 8 - 7 6 5 4 - 3 2 1 0
; o o i o i i o o i i i i i i i i
;
    XCODE_PORT_B_M_PBC    ;set B control register for general IO
    XCODE_PORT_B_M_PBD    ;set the default outputs

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 117 -

```

        XCODE_PORT_B_M_PBDOR      ;set B register direction
; initialize the host interrupt vector
        INIT_HOST_VECTORS_CD
restart
; set the interrupt for host interrupts
; HOST set to IPL 2

        movep    #>S080C,x:<<M_IPR      ;set int priorities and edges
        andi     #Sfc,mr                ;turn on the interrupt system

        cri      #S33,mr
        nop
        nop
        nop

; clear the analog to digital converter to restart calibration
        CLR_ADC_RESET

; disable the ancillary data received interrupt

        bclr     #M_RIE,x:<<M_SCR

        move      #>OFF_LEDS_CD,b        ;initialize leds as off
        move      b,y:<<word_out

;*****
; TEST NOTICE THAT THE FOLLOWING DATA IS ENCODED AND PUT INTO A HIGH MEMORY
; AND WILL BE CHCKED WOTH THE CODED DATA ALL THE TIME WHILE THE PROGRAM
; RUNS TO MAKE SURE THAT NONE OF A WORD IS IN ERROR
; TEST DATA

; initialize the buffer to be encoded for testing

        OFF_REED_SOL_LED_CD
        move      #framebuf,r0           ;indicate no problem with Reed Solomon
        clr       a                      ;code the 1st of the encoded frames
        clr       a                      ;zero the test value accumulator
        clr       a                      ; & to increment in the test buffer

; set the frame buffer to sequentially incremented values

        do        #96,_init1
        add       x0,a
        move      a1,x:(r0)+
_init1:

; do the reed solomon encoding on the test frame buffer

        move      #framebuf,r0           ;i/p pointer of buffer to be RS-CODED
        move      #Sbf,m0               ;frame buffer is circular - 2 frames
        move      #reedsolbuf,r1        ;o/p pointer for CODED data to be stored
        jsr       <new_rs               ;encode via reed solomon

; test if the reed solomon codec worked or NOT

```



- 118 -

```

move    #reedsolbuf,r0      ;o/p pointer for CODED data to be stored
move    #RStest,r1          ;pointer for the verification table

;verify that the reed solomon coded values are correct

do      #96, RS_Chk
move    x:(r0)+,x0          ;Get current coded data output
move    x:(r1)+,a           ;Get precoded look up table value
cmp     x0,a                ;compare 2 values
jeq     < Same              ;If SAME No problem
ON_REED_SOL_LED_CD         ;indicate no problem with Reed Solomon
enddo
nop

_Same   nop

_RS_Chk

ON_ALARM_LED_CD             ;light alarm led indicator
TST_SET_ALARM_RELAY_CD,_set_led_0 ;unless already set,
SET_ALARM_RELAY_CD         ;set the alarm relay line on

_set_led_0
SET_LEDS_CD
INTERRUPT_HOST_CD          ;inform the host

; Clear all of the y memory

clr     a                   ;value to set x memory to
move    #ffff,m0            ;just in case, set to linear buffer
move    #startyl,r0         ;set starting address low y-memory
move    #(endyli-startyli),r1 ;set loop count
rep     r1                  ;clear it
move    a,y:(r0)+           ;set starting address high y-memory
move    #startyhe,r0        ;set loop count
move    #(endyhe-startyhe),r1 ;set loop count
rep     r1                  ;clear it
move    a,y:(r0)+

;set linear buffer control

move    m0,y:<linear

;set the CRC-16 protection checksum as applicable and set the
; CRC-16 checksum mono frame bit count for the old ISO method:
; a. header bits covered by any type of frame
;    plus bits for the left channel also apply to any type of frame
; b. save old ISO bit count for this frame

bset    #PROTECT,y:<stereoc ;checksum protection applies (1=YES)
move    #>CRC_BITS_A+CRC_BITS_B,a ;header plus one channel bits
move    a,x:CRCold          ;set the old ISO CRC-16 bit count

;check the switches to determine bit rate and framing type
;get the external switches to determine:
; PCM input data sampling rate
; type of audio compression to format for output (MUSICAM/G722)
; if MUSICAM, the frame bit rate
; if MUSICAM, ancillary data baud rate

GET_SWITCHES_CD gsws_00

```

- 119 -

```

jsr      <getsws

move     x:tstsmpl,y1
move     y1:y:smplrte      ;set PCM data sampling rate code
move     x:tstfrme,y1
move     y1:y:frmttype     ;set type of frame (mono) to code
move     x:tstband,y1
move     y1:y:bnwidth      ;set bit allocation sub-band width code
move     x:tstcode,y1
move     y1:y:oputcode     ;type of encoded output (MUSICAM/G722)
move     x:tstrate,y1
move     y1:y:frmrte       ;set the frame rate i/p code
move     x:tstbaud,y1
move     y1:y:baudrte      ;set ancillary data baud rate code
move     x:tstoccs,y1
move     y1:y:<oldccs      ;set MPEG-ISO vs old CCS CDQ100's

;set framing mode led

move     y:frmttype,x0
move     x0,y:cpfrtyp      ;set current frame type
                           ;set current frame type for output to

;indicate mono framing (only frame type supported)

bset     #STEREO_vs_MONO,y:<stereo

;based on sample rate (low or high) set the addresses for various tables:

move     y:smplrte,b
tst      b
jne      <_hi_tables

move     #b_ilo,r0          ;address of b_i table for low rate
move     #fmaplo,r1         ;address of fmap table for low rate
move     #ThrsSLBlo,r2      ;address of ThresSLB table for low rate
move     #Thrhldlo,r3       ;address of Threshld table for low rate
move     #cblo,r4           ;address of cb table for low rate
move     #g_cblo,r5         ;address of g_cb table for low rate

;indicate coding at low sampling rate for compression

bclr     #LOW_vs_HIGH_SAMPLING,y:<stereo

jmp      <_set_tables

_hi_tables
move     #b_ilo,r0          ;address of b_i table for high rate
move     #fmaplo,r1         ;address of fmap table for high rate
move     #ThrsSLBlo,r2      ;address of ThresSLB table for high rate
move     #Thrhldlo,r3       ;address of Threshld table for high rate
move     #cblo,r4           ;address of cb table for high rate
move     #g_cblo,r5         ;address of g_cb table for high rate

;indicate coding at high sampling rate for compression

bset     #LOW_vs_HIGH_SAMPLING,y:<stereo

_set_tables
move     r0,y:b_i           ;set addr of b_i table selected
move     r1,y:fmap          ;set addr of fmap table selected

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



- 120 -

```

move    r2,y:ThresSLB          ;set addr of ThresSLB table selected
move    r3,y:Threshld          ;set addr of Threshld table selected
move    r4,y:cb                ;set addr of cb table selected
move    r5,y:g_cb              ;set addr of g_cb table selected
move    #DbAddTbl_6db,r3
move    r3,y:dbaddtbl

;based on the sampling rate and framing bit rate selected:
;set the sampling rate code for the ISO frame header
;set the framing bit rate code for the ISO frame header
;set the frame size in words and bits
;set the applicable bit allocation control parameters

move    #sampling,r0           ;addr of sampling rate codes
move    y:smpirte,b            ;offset to sampling code table
tst     b                      ;test for sampling rate of zero
; & set register to advance thru table
; if code is zero, we're there
jeq     <_smpicds_

rep     b
move    (r0)+n0                ;position to selected sampling rate code

_smpicds_
move    y:(r0)+,x0             ;get ISO frame header sampling code
move    x0,y:smpicde           ;save ISO code to encode in frame header
move    y:(r0)+,x0             ;get ISO frame header id bit
move    x0,y:smpidbit          ;set ISO frame header id bit
move    y:(r0)+,x0             ;get mono channel MAXSUBBANDS
move    x0,y:<maxsubs          ;set working MAXSUBBANDS
move    (r0)+                  ;step over dual channel MAXSUBBANDS
move    #4,n0                  ;in case of MPEG-ISO
move    #0,y:<cmprset1         ;CCS compression is not applicable
bclr    #0,y:<oldccs,_smploffs_ ;if MPEG-ISO, skip over old CDQ1000's
jclr

;encoding old CCS CDQ1000
move    y:(r0)+,x0             ;old CDQ1000 frame header sampling code
move    #smpidbit,r1           ;to check ISO frame header id bit
move    x0,y:smpicde           ;save old code to encode in frame header
jset    #0,y:(r1),_no_compress_ ;if ISO high sampling, no compression
bset    #0,y:<cmprset1         ;do CCS compression encoding

_no_compress_
move    y:(r0)+,x0             ;get old CDQ1000 frame header id bit
move    x0,y:smpidbit          ;set ISO frame header id bit
move    y:(r0)+,x0             ;get mono channel MAXSUBBANDS
move    x0,y:<maxsubs          ;set working MAXSUBBANDS
move    (r0)+                  ;step over dual channel MAXSUBBANDS
jmp     <_aftscds_             ;continue

_smploffs_

;MPEG-ISO encoding
move    (r0)+n0                ;skip over old CCS CDQ1000 values

_aftscds_
move    y:(r0)+,x0             ;get MAXCRITBND value @ sample rate
move    x0,y:<maxcritbnds      ;set MAXCRITBND at selected sampling
move    y:(r0),x0              ;get NMSKFREQS value @ sample rate

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

- 121 -

```

move    x0,y:<nmskfreqs      ;set NMSKFREQS at selected sampling
move    y:frmrate,b         ;test bit rate to set audio data size
move    #bitrates,r0        ;addr of framing bit rate info
tst     b                    ;test for rate of zero
; & set register to advance thru table
; if code is zero, we're there
jeq     <_bit_offs_

rep     b
move    (r0)+n0              ;position to selected bit rate code

_bit_offs_
;set the table offset based on sampling rate

move    y:smplrte,b         ;get the sample rate code
tst     b                    ;test if low sampling rate
; & set offset to proper sampling rate
; if low rate, addr is set
jeq     _bit_smp1_

rep     b
move    (r0)-n0              ;position to selected sample rate

_bit_smp1_
jclr    #0,y:<oldccs_bit_cds_ ;if MPEG-ISO, continue
move    (r0)-                ;adv to old CCS CDQ1000's code

_bit_cds_
move    y:(r0)+.n1          ;get bit rate code for frame header
jset    #0,y:<oldccs_aftbcd_  ;if old CCS CDQ1000's, continue
move    (r0)-                ;skip over old CCS CDQ1000 code

_aftbcd_
move    y:(r0)+.y1          ;selected bit rate frame size in words
move    y:(r0).r2           ;number of audio bits in an output frame

move    n1,y:bitrate         ;audio bit rate code for frame hdr
move    y1,y:<outmus          ;set # of words in a frame
move    r2,y:frmbits         ;musicam audio portion of frame

;set bandwidths based on sampling rate, bit rate and band width selection

move    y:smplrte,b         ;set bandwidths based on sampling rate
move    y:frmrate,a         ;set bandwidths based on frame bit rate

jsr     <bandwidth

move    y:z3_psych,a         ;get the selected sub-bands, if any
move    a,y:<usedsb          ;set initial used sub-band value
move    #>MINSUBBANDS_CCS,x0 ;set minimum sub-bands to be used
cmp     x0,a                 ;see if subs is too small
; & set default value of maximum
jlt     <default_used_00     ;if less, default the used sub-bands
cmp     x0,a                 ;see if less than maximum sub-bands
jlt     <_after_used_00     ;if less, we're ok

_default_used_00
;default: the used sub-bands to max sub-bands

move    x0,y:<usedsb

```



- 122 -

_after_used_00

; calculate buffer length controls

```

move    #>2,x1
mpy     x1,y1,a #>1,x1      ;set the mod buffer for 2 frames
asr     a                    ;align integer result
move    a0,a                ;shift integer result
sub     x1,a                ;(frame numb words * 2) - 1

```

; now save the above buffer control values

```

move    a1,y:<outsize      ;set circular buffer ctrl for o/p buffer

```

; set the type of stereo intensity code as nominal 4 subbands (not applicable)

```

move    #>INTENSITY_4,x0    ;stereo intensity code for default of 4
move    x0,y:stintns        ;save for frame header info

```

; Set output write read pointer to something safe since interrupts will be on before it is set properly.

```

move    #framebuf,r0        ;address of output encoded frames buffer
move    r0,y:<oprptr        ;set the output read buffer

```

; set up for ancillary data to be decoded from a framed and transmit via rs232

```

; a. zero the input data byte counter and bytes for current frame
; b. set address of clock table, baudclk, based on baud rate (0 thru 7)
; c. set table offset by baud rate;
;    these are standard CDQ2000 set by macro, BAUDCLK, in box_ctl.asm:
;    0 = 300 baud
;    1 = 1200 baud
;    2 = 2400 baud
;    3 = 3200 baud
;    4 = 4800 baud
;    5 = 38400 baud
;    6 = 9600 baud
;    7 = 19200 baud
; d. set transmit enable (for xon/xoff)
; e. get and set the clock for baud rate from the table
; f. get and set the max bytes for baud rate from the table
; g. set the data input and output pointers
; h. set receive enable
; i. set receive enable interrupt

```

```

move    #0,x0                ;zero the received data counter
move    x0,y:<bytecnt        ;zero the byte counter
move    x0,y:<bytesfrm       ;zero the current frame byte counter
move    #baudclk,r0          ;get data baud rate table address
move    y:baudrte,b          ;set to access clock at baud rate
tst     b                    ;test for rate of zero
; & set register to advance thru table
; if code is zero, we're there
jeq     <_baudrte_
rep     b
move    (r0)+,n0             ;position to selected baud rate code

```

```

_baudrte_ move    y:r0,r2    ;get clock value at baud rate

```

BAD ORIGINAL

-123-

```

move    y:smplrte,n0          ;now get sampling rate offset
move    #databytes,x0        ;get addr of the data byte buffer
move    y:(r0+n0),n1         ;get max byte count at sampling rate
move    n1,y:maxbytes        ;store maxbytes for scixmt to check
move    x0,y:<dataiptr        ;address for next byte received
move    x0,y:<dataoptr        ;addr for next byte to output to frame
movep    r2,x:<<M_SCCR        ;set the clock for selected baud rate
bset    #M_RE,x:<<M_SCR       ;set receive enable
bset    #M_RIE,x:<<M_SCR      ;data expected set receive interrupt
bset    #M_TE,x:<<M_SCR       ;set transmit enable

;enable the host command interrupt

bset    #M_HCIE,x:<<M_HCR

; Set and clear a flag so we can set the scope trigger

ON BITALLCC_LED_CD           ;set a different flag for debug
OFF_BITALLCC_LED_CD

; Now form the two pointers to the output buffer.
; frmstrt is the write pointer and frmnext is the read pointer.
; frmstrt is used to point to where the current buffer is for outputting
; data into. This data is a result of the current musicam coding.
; frmnext is used to point to the address for outputting of data
; to the external device.

move    #framebuf,r0          ;address of the output frame buffer
move    y:<outmus,n0           ;set the output read ptr
move    y:<outsize,m0          ;set the output buffer circular ctl
move    r0,y:<frmstrt          ;1st frame at start of buffer
move    (r0)+n0               ;advance to start of 2nd frame
move    r0,y:<oprptr           ;set the output read buffer
move    r0,y:<frmnext          ;set the next frame to write into
move    (r0)-                 ;set up last word addr of curr frame
move    r0,y:<frmlast          ;for block sequence numbering
move    y:<linear,m0           ;reset to linear buffer

;set number of fixed bits required, and the number of available bits for audio

jsr     <bitpool

move    x0,y:fixbits           ;save fixed bit count
move    x1,y:audbits           ;save bit count available for alloc

;initialize for receiving data for xpsycho routines

move    #inpcm,r0              ;get the input pcm data buffer
move    r0,y:<ipwptr           ;set start address for input pcm data
move    #xbuf,r0               ;set starting position in x buffer
jsr     <polyaini              ;init the poly analysis filter

; IRQA set to IPL 3, negative edge (lowest priority)
; SSI set to IPL 3
; IRQB set to IPL 3, negative edge (highest priority)
; HOST set to IPL 2
; SCI set to IPL 3

movep    #>$f83f,x:<<M_IPR      ;set int priorities and edges

```

-124-

```

;wait for the dust to settle before pushing onward
;
;   move    #>XCODE_STARTUP,a
;   jsr     <wait
;
;   SET_ADC_RESET                      ;stop A to D calibration
;
;test MUSICAM versus G722:
;   if MUSICAM, go to the TOP of frame processing
;   if G722, jump to that routine and restart upon return
;
;   move    y:opcode,a                ;MUSICAM vs G722
;   tst     a                          ;if zero,
;   jeq     <go_on_                    ;it's MUSICAM, enter that loop
;   jsr     <g722                      ;handle G722
;
;G722 output selected, boot up XMCRG722 from the low portion of chip
;
;   bclr    #11,x:<<M_PBD              ;clr boot c000 for XMCRG722 boot (0000)
;   jmp     <bootup                    ;boot in XMCRG722
;   jmp     <restart                   ;restart with new switches
;
;_go_on_
;handle MUSICAM encoding
;
;   andi    #$fc,mr                   ;turn on the interrupt system
;
;main loop thru the frames of data set up by the left and right
; xpsycho dsp for bit allocation and framing by the xcode dsp
;
top
;!!!dbg
;   nop
;   nop
;   move    y:dbgcnt,a
;   move    #>1,x0
;   add     x0,a
;   move    a,y:dbgcnt
;   jmp     <_init1
;!!!dbg
;
;!!!dgst      bset    WATCH_DOG          ;tickle the dog
;!!!dgst      bclr    WATCH_DOG          ;tickle the dog
;              TOGGLE_WATCH_DOG_CD
;
;get the external switches to determine if any changes that signal a restart
;
;   GET_SWITCHES_CD gsws_10
;   jsr     <getsws
;   jclr    #4,y:<not_appl,_lets_go ;!!!debug - remove for normal
;
;test MUSICAM versus G722:
;   if G722, jump to restart
;   if MUSICAM, continue
;
;   move    x:tstcode,a                ;MUSICAM vs G722
;   tst     a                          ;if zero, it's MUSICAM
;   jne     <restart                   ;it's G722, start over to boot

```



-125-

;!!!2/8/93

TST_SET_G722_DATA_CD.restart

;!!!2/8/93

;we have to restart with new framing criteria.

; protect the decoding of frames by clearing 2 successive frame

```

move    y:<frmstrt,r6          ;set starting for output buffer
move    y:<outsiz,m6           ;set the output buffer circular cti
clr     a
do      y:<outmus,_clear_1      ;clear the 1st frame
move    a,x:(r6)+

```

_clear_1

;!!!2/8/93

TST_SET_G722_DATA_CD.restart

;!!!2/8/93

```

jclr    #0,y:<timer,_clear_1   ;check for new frame
bclr    #0,y:<timer
move    y:<frmnext,r6          ;set starting for output buffer
do      y:<outmus,_clear_2      ;clear the 2nd frame
move    a,x:(r6)+

```

_clear_2

;!!!2/8/93

TST_SET_G722_DATA_CD.restart

;!!!2/8/93

```

jclr    #0,y:<timer,_clear_2   ;check for new frame
bclr    #0,y:<timer
move    y:<linear,m6           ;restore to linear buffer control
jmp     <restart               ;let's start anew

```

_lets_go

;initialize stereo control settings to reflect current transmission

```

jsr     <setctls
jclr    #0,y:<timer,top         ;check for new frame
bclr    #0,y:<timer
bclr    #0,y:<qtalloc           ;clr 0.024/0.036 msec timer bit alloc

```

;now set the used sub-bands for this frame

```

move    y:z3_psych,a          ;get the selected sub-bands, if any
move    a,y:<usedsb            ;set initial used sub-band value
move    #>MINSUBBANDS_CCS,x0   ;set minimum sub-bands to be used
cmp     x0,a #>MAXSUBBANDS_CCS,x0 ;see if subs is too small
; & set default value of maximum
jlt     <_default_used_10      ;if less, default the used sub-bands
cmp     x0,a                   ;see if less than maximum sub-bands
jlt     <_after_used_10        ;if less, we're ok

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-126-

_default_used_10

;default the used sub-bands to max sub-bands

move x0,y:<usedsb

_after_used_10

;set the CCS compression as per control parameter (n_psych)

```

bclr    #0,y:<cmprsc1
move    y:n_psych,a
move    #.5,x0
cmp     x0,a
jlt     <_no_compress
bset    #0,y:<cmprsc1

```

```

;default as do not use CCS compression
;get the parameter from the table
;if less than .5, no CCS compress
;see if use CCS compression or not
;if less, do not use CCS compression
;otherwise, set flag to use CCS compress

```

_no_compress

;the new data for the next frame is all set, lets do it

jsr <doframe

INTERRUPT_HOST_CD

;inform the host

;pass the MUSICAM encoded frame off for reed solomon encoding

```

move    y:<frmstrt,r0
move    y:<outsize,m0
move    #reedsolbuf,r1
jsr     <new_rs

```

```

;set starting for output buffer
;set the output buffer circular c1
;set starting for output buffer
;call Reed Solomon encoding routine

```

;!!!dbg

jmp <top ;!!!dbg: skip Reed Solomon

;!!!dbg

;copy the reed solomon encode frame into the output frames buffer

```

move    y:<frmstrt,r0
move    y:<outsize,m0
move    #reedsolbuf,r1
do      y:<outmus,_copy_rs
move    x:(r1)+,x0
move    x0,x:(r0)-

```

```

;set starting for output buffer
;set the output buffer circular c1
;set starting for output buffer

```

_copy_rs

jmp <top

end start

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-127-

```

opt      fc,mex

; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \URDCDSYN\autosmpl.asm: modified to coordinate with BEN's mux

      title  'Decoder Auto Determine Sampling Rate'

; This routine attempts to determine the sampling rate of MUSICAM frame of
; input data being fed to a MUSICAM decoder. It tries to match on the
; selected bit rate a corresponding sampling rate that are predefined for
; the given units capabilities.

;on entry:
; y:frmrate = indicates which bit rate was selected
; y:<ctlglgs = NO_LINES bit is set as to whether split frames possible
; x:maxtries = the number of attempts at framing that should be made
;               before determining that the input data is not MUSICAM

      include 'def.asm'
      include '..\common\ioequ.asm'
      include 'box_ctl.asm'
      include 'box_smpl.asm'
      include 'box_tbis.asm'

      section highmisc
      xdef  syncptrn

      org    yhe:
stauto_yhe

      syncptrn      ds      4           ;4 possible sync & hdr patterns

endauto_yhe
      endsec

      section lowmisc
      xdef  synccnt
      xdef  syncmtch
      xdef  syncwrds
      xdef  syncbits
      xdef  syncfrms
      xdef  synced

      org    yli:
stauto_yli

      synccnt      ds      1           ;count of sync patterns to check
      syncmtch     ds      1           ;pattern matched (odd=padded)
      syncwrds     ds      1           ;words per frame (if pad diff -1)
      syncbits     ds      1           ;bit offset to frame start
      syncfrms     ds      1           ;number of frame to sync up on
      synced       ds      1           ;count of frames sync'ed

endauto_yli
      endsec

      section highmisc
;!!!BEN      xdef  srchrte              ;!!!BEN

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-128-

```

xdef srchtries      ;!!!BEN
;!!!BEN
xdef maxtries
xdef tstsmpl
xdef fndbit
xdef fndsmpl
xdef fndidbit
xdef padbit
xdef sampletable

org xhe:
stautc_xhe

;!!!BEN
srchrate      dc      0      ;index to rates in sample rate table
srchtries     dc      0      ;failure counter of auto sample attempts
;!!!BEN
maxtries      dc      0      ;current auto determine max tries
tstsmpl       dc      0      ;sample code under test
fndbit        dc      0      ;bit rate code from frame header
fndsmpl       dc      0      ;verify found sampling rate selection
fndidbit      dc      0      ;verify found sampling rate id bit
padbit        dc      0      ;save padding bit from the header

        SAMPLETABLE      ;table for sample rate auto determination

endauc_xhe
endsec
org phe:

autosample

        CLR_DAC_RESET      ;clear the DAC reset line to mute output

;!!!BEN
;turn off the interrupt system
;
;      ori      #S03,mr
;
; Now set priorities of the IRQA and SSI peripherals
; IRQA priority = 2
; IRQB priority = 3
; SSI  priority = 2
; SCI  priority = 2
;
;      movep    #>Sa03e,x:<<M_IPR      ;set int priorities and edges
;!!!BEN

_auto_AA

        jset     #AUTONEXTFRAME,y:<process,_auto_continue

;build up the frame length table based on the selected bit rate

        move     #sampletable,r0      ;addr of sample rate frame lengths
        move     #AUTOBYSAMPLE,nC     ;set auto sample offset to next rate
        move     x:srchrate,b         ;get next rate index to search for
        tst      b                    ;see if 1st sample rate in table
        jeq      _auto_BB             ;if so, skip address adjustment

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

129

```

do      b, _auto_BB      ;for index count, adj table addr
move    (r0)+n0          ;advance to next sample rate

_auto_BB

;!!!BEN
;for the number of sampling rates supported, set table of frame lengths
;
do      #NUMSAMPLERATES, _auto_900
;7/12/94: test sampling rate as not applicable to current project
;
move    r0,y:<svereg      ;save current table address
move    x:(r0)+,b        ;get rate applicable code (0 = APPLIES)
bclr    #1,y:oldccs      ;clear y:oldccs frames CDQ1000 flag
tst     b                ;see if not applicable (-1 = N/A)
jlt     _auto_800        ;if N/A, go to try next sampling rate

;now test for framing on old CDQ1000 low sampling rate old frames
;
jeq     _auto_A          ;if zero, not old ccs CDQ1000 frames
bset    #0,y:oldccs      ;indicate old CCS
bset    #1,y:oldccs      ;indicate old CDQ1000 frames
bset    #DECOMPRESS_PACKED,y:<ctrlflgs ;handle CCS compression

_auto_A

;get the MUSICAM frame header ID bit that indicates high vs low sampling rates
;
move     x:(r0)+,x0      ;get the high/low rate hdr id bit
move     x0,y:smplidbit  ;save for translate rate code
move     r0,r1           ;address of entries at sample rate

;translate the raw bit rate code to the internal rate index code
; based on whether the sampling rate is high (y:smplidbit 1=high) or low (0)
;and validate that the rate is supported by the software and/or hardware
;
move     #translatables,r0 ;addr of the translation table
move     y:rawrate,n0      ;to offset to translated index
nop
move     (r0)+n0           ;pos to bit rate translate 1st value
move     (r0)+n0           ;pos to bit rate translate 2nd value
move     y:smplidbit,n0    ;low (0) or high (1) sample rate select
move     #>-1,a            ;to see if not supported
move     y:(r0+n0),x0      ;get the translated rate index code
move     x0,a             ;see if not supported rate
cmp      x0,a              ;not supported, try next sampling rate
jeq      _auto_800

;set the supported framing bit rate table index code
;
move     x0,y:frmrate      ;bit rate index code

;set up the framing patterns table at sampling rate/framing bit rate
;
move     #AUTOBYBITRATE,n1 ;numb parameters per bit rate
move     y:frmrate,b       ;get the defined bit rate
tst      b                 ;test if code zero
; & set table sample rate code
;
jeq      _auto_00         ;if zero, skip addr adjustment
rep      b                 ;position to selected bit rate

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-130-

```

move    (r1)+n1

_auto_00
move    x0,x:tstsmpl    ;save sample rate code
;build up the table of framing patterns at this sample/bit rate
move    #syncptrn,r2    ;table of framing patterns to match
;set at least the 1st two patterns: unpadded and padded (possibly)
move    x:(r1)+,b        ;get 1st defined framing pattern
tst     b,b,x0           ;if 1st pattern is zero, not valid
; & save 1st defined framing pattern
;bit rate not supported @ sample rate
;insert the pattern in test table
jeq     _auto_800        ;get 2nd defined framing pattern
move    x0,y:(r2)+       ;if pattern zero (NO padding possible)
move    x:(r1)+,b        ; & set pattern count to 1 (at least)
tst     b,#>1,x1         ;if zero, use 1st pattern over again
;else, use the padded framing pattern
;set pattern count to 2
jeq     _auto_10
move    b,x0
move    #>2,x1

_auto_10
move    x0,y:(r2)+       ;insert 2nd pattern in test table
;now if split mono framing is possible, set up to look for those frames
jclr    #NO_LINES,y:<ctlflgs,_auto_20 ; NOT appl if one or both lines
move    x:(r1)+,b        ;get 3rd defined framing pattern
tst     b,b,x0           ;if pattern zero (NOT split frames)
; & in case of duplication as 4th
;if zero, NOT eligible for split frames
jeq     _auto_20        ;insert 3rd pattern in test table
move    x0,y:(r2)+       ;get 2nd defined framing pattern
move    x:(r1)+,b        ;if pattern zero (NO padding possible)
tst     b,#>3,x1         ; & set pattern count to 3
;if zero, use 1st pattern over again
;else, use the padded framing pattern
;set pattern count to 4
jeq     _auto_20
move    b,x0
move    #>4,x1
move    x0,y:(r2)+       ;insert 4th pattern in test table

_auto_20
;set count of framing patterns inserted in the framind pattern table
move    x1,y:<synccnt    ;set the pattern count for framing
;get the frame length values at this sample/bit rate
move    #framevalues,r0  ;addr of sample rate values
move    #FRAMEBYSAMPLE,n0 ;numb parameters per sample rate
move    x:tstsmpl,b      ;to see if need to adjust address
tst     b                ;if code 0, no need to shift address
jeq     _auto_40         ;if 0, get the 3 parameters
;adjust the table address to proper sampling rate parameters
rep     b
move    (r0)+n0

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-131-

```

_auto_40
    move    #FRAMEBYBITRATE,n0      ;numb parameters per framing bit rate
    move    y:frmrate,b             ;test bit rate to set audio data size
    tst     b                       ;if code 0, no need to shift address
    jeq     _auto_50                ;if 0, get the parameters

;adjust the table address to proper framing bit rate parameters at sample rate

    rep     b
    move    (r0)+n0

_auto_50
    move    y:(r0)-,r1              ;get the words per frame at rate
    move    r1,n1                   ;to calc circular doubled buffer cti
    move    (r0)+                   ;skip the bit count per frame
    move    (r1)+n1                  ;double framing buffer
    move    (r1)-                    ;for circular double buffer cti
    move    r1,y:frmmod              ;save framing circ buffer cti
    move    y:(r0)-,b               ;get any padded frames DIFF value
    tst     b                        ;to see if word count adj needed
                                ; & restore frame length in words

    jeq     _auto_60
    move    (r1)-                   ;decrement word count if padded

_auto_60
    move    r1,y:<syncwrds            ;set the words per unpadded frame
    move    y:(r0)-,x0              ;get any unpadded frame extra bits
    move    x0,y:<syncbits           ;set any unpadded frame extra bits
    move    #0,r3                   ;to zero the failure counter
    move    r3,x:srchtries           ;zero the failure counter
    bclr    #0,y:<protect            ;start looking for CRC protection
    bclr    #0,y:privacybit         ;start looking for privacy bit off

_auto_70
    ;;;BEN
    ;;;turn off the interrupt system
    ;;;
    ori     #503,mr

;initialize for the interrupt routine to try to frame

    move    x:srchtries,r3          ;current failuer counter
    move    #0,x0                   ;clear all bits
    move    (r3)+                   ;increment attempt ctr
    move    r3,x:srchtries          ;save incrment failure counter
    move    x0,y:<inpstat            ;flags to control i/p routine
    bset    #2,y:<inpstat            ;flag to do pad framing
    ;;;
    move    y:frmmod,a0             ;for framing buffer size
    ;;;
    move    a0,y:<inpsize            ;store for ssirec rtn to store
    move    #>AUTO_FRAMES,y1        ;# of frames to match
    move    y1,y:<syncfrms           ;set number of frames to sync
    move    x0,y:<syncd              ;zero the synced frame counter
    ;;;
    move    #syncbuf,x0             ;address of the input buffer
    ;;;
    move    x0,y:<inwptr            ;set the input write pointer

    ;;;BEN
    ;;;before turning on the interrupts, restart the input data stream process
    ;;; that inputs bits to form 24-bit words

```



-132-

```

;;      move      #Bit1T6In,r7          ;init the bit input buffer ptr
;;      andi      #Sfc,mr              ;turn on the interrupt system
;;
;;hang out here until framed or failed
;;
;;_auto_80
;;      bset      WATCH_DOG            ;tickle the dog
;;      bclr      WATCH_DOG            ;tickle the dog
;;
;;      bset      #AUTONEXTFRAME,y:<process
;;
;;!!!BEN:perform old ssirec auto sampling on current frame
;;
;;_auto_continue
;;
;;we are now attempting to frame:
;;if start of "syncing" (bit 3 not set),
;;  set 1st word of pair to check
;;  set starting word offset
;;  set flag to set 2nd word
;;  continue to react when 2nd word to check comes in
;;else,
;;  see if waiting for the 2nd word or counting looking for the next sync
;;
;;      move      y:frmcrr,r4          ;set start of the frame addr
;;      move      y:frmemod,m4         ;set circular buffer 2 frames
;;
;;_auto_CC
;;
;;start looking for framing pattern
;;
;;      jset      #3,y:<inpstat,_auto_35 ;we have set the 1st word, continue
;;      clr       a                    ;init for the 2 words to check
;;                      r4,y:wrdooff    ; & save initial start word offset
;;
;;      move      x:(r4)+,a1           ;set 1st word to check (incr write ptr)
;;      bset      #3,y:<inpstat         ;flag to check the 2nd word
;;      move      #0,r2                ;start count of words looking for sync
;;      jmp       _auto_CC             ;try 2nd word
;;
;;if waiting for 2nd word to check (bit 4 not set),
;;  put new word in a0 to look for the 24 bit pattern
;;  start the bit offset counter
;;  loop through 24 bits over 1st and 2nd word trying to match one
;;  of the defined sync patterns
;;else,
;;  we found a pattern and are trying sync up on the next frame
;;
;;_auto_35
;;      jset      #4,y:<inpstat,_auto_105 ;counting to check next frame sync
;;      move      x:(r4),a0            ;set the 2nd word to search
;;      move      #0,r1                ;init the bit offset counter
;;      do        #24,_auto_65
;;
;;see if current offset contains a valid sync pattern
;;
;;      move      a1,b                  ;current bit offset pattern
;;      move      #syncptrn,n0          ;addr of array of sync patterns
;;      move      #0,r0                ;offset to 1st pattern

```



-133-

;loop through the available sync patterns

```

do      y:<syncnt, _auto_55
move    y:(r0+n0),x0      ;get the next sync pattern to check
cmp     x0,b              ;see if pattern matches
jne     _auto_45          ;if not, try next pattern

```

;we found a framing pattern, set the indication and break out to proceed

```

bset     #4,y:<inpstat      ;indicate the match
enddo
enddo
jmp      _auto_65          ;we matched the pattern

```

_auto_45

;try the next framing pattern

```

move     (r0)+

```

_auto_55

;try the next bit for a match of a framing pattern

```

asl      a      (r1)+      ;shift left into a1
                        ; & increment the bit shift counter

```

_auto_65

;if the pattern was not matched,

; set the next word as the offset

; increment the address for the next word

; exit the interrupt routine and wait for a new 2nd word to check

```

clr      a      (r2)+      ;zero the sync'd frames counter
                        ; & incr count of words looking for sync
jset     #4,y:<inpstat,_auto_75 ;if match, set up to check next frame
move     y:<syncwrds,a      ;get number of words per frame
move     #>FRAME_OVERAGE,x0 ;to add some cushion to frame length
add      x0,a      r2,x0    ;add cushion to frame length
                        ; & get words checked so far
cmp      x0,a      r4,y:wrdoftest more than frame checked for sync
                        ; & save possible new start word offset

```

;if more than a full frame has been searched without finding SYNC:

; we failed at framing at this sampling/bit rate

```

jlt      _auto_155          ;indicate failure at sample/bit rate
move     x:(r4)+,a1         ;set new 1st word to check (incr ptr)
jmp      _auto_CC          ;try new 2nd word

```

_auto_75

;frame matches a sync pattern:

; update the sync'd frame counter

; save the sync pattern match index to test for padding or not

; store the new bit offset to start this frame

; set the address and offset for the next frame

; see if padding needed,

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-134-

```

move    a,y:<synced                ;update the sync'd frame counter
move    r0,y:<syncmtch             ;save matched pattern index
move    r1,y:bitoff               ;save the bit offset
move    y:wrdoft,r0              ;address start last frame
move    y:frmmod,m0              ;set circular buffer
move    y:<syncwrds,n0            ;words to next frame
move    y:bitoff,a               ;get the bit offset start
move    (r0)+n0                  ;address for next frame start
move    y:<syncbits,x0            ;get unpadded frame extra bits
add     x0,a    #>PAD_SLOT,x0    ;add extra bits to offset
                                           ; & set up for any needed padding
jclr    #0,y:<syncmtch,_auto_85   ;match index even, NOT padded
add     x0,a                    ;add the padded bits

_auto_85
;see if bits exceeds full word and adjust

move    #>24,x0                  ;24 bits per word
cmp     x0,a                    ;see if next address needed
jlt     _auto_95                ;if offset within word, continue
sub     x0,a    (r0)+           ;adjust the bit offset by full word
                                           ; & increment the start address

_auto_95
;set address and bit offset to match the next frame

move    r0,y:wrdoft              ;start next frame word address
move    a,y:bitoff               ;start next frame bit offset
move    (r4)+                   ;advance the write pointer
move    y:linear,m4              ;restore as a linear buffer
move    y:linear,m0              ;restore as a linear buffer
bclr    #5,y:<inpstat             ;clear reached frame indicator
rts                                           ;BEN - exit rtn and wait for next frame

_auto_105
;if ready to check the new frame as it comes in
; test if expected frame start address has been reached
; if so, set indicator to check the next word received (2nd in the frame)
; otherwise, keep accepting frame words into buffer
;else,
; check for the pattern in the 1st and 2nd word (latest received)

jset    #5,y:<inpstat,_auto_115   ;to test if frame start addr hit
move    r4,x0                   ;address to match
move    y:wrdoft,a              ;see if address hit
cmp     x0,a    (r4)+           ; & increment the write pointer
jne     _auto_155               ;if not, frame length problem

;we have the 1st word of the frame
; set indicator to check 2nd word for framing pattern

bset    #5,y:<inpstat            ;indicate check next word for pattern
jmp     _auto_CC                ;to check 2nd word

_auto_115

```

SUBSTITUTE SHEET (RULE 26)

-135-

;we now have the 2 words to check this frame for framing

```

clr      a      #>1,x1      ;clear the register to align pattern
                                ; & set to increment frame match count
move     x:(r4)-,a0          ;retrieve 2nd word (back up to 1st)
move     x:(r4)+,a1          ;retrieve 1st word (forward to 2nd)

```

;if a bit offset, shift over the expected bits to align the pattern

```

move     y:bitoff,b          ;to see if a shift is needed
tst      b                  ;see if zero
jeq      _auto_125          ;if so, skip the shift

```

;shift left to align pattern in a1

```

do       b,_auto_125
asl      a

```

_auto_125

;see if current offset contains a valid sync pattern

```

move     a1,b                ;to test shifted pattern from frame
move     #syncptrn,n0        ;addr of array of sync patterns
move     #0,r0                ;offset to 1st pattern
bclr     #6,y:<inpstat        ;indicate no match yet

```

;loop through the available sync patterns

```

do       y:<synccnt,_auto_145
move     y:(r0+n0),x0        ;get the next sync pattern to check
cmp      x0,b                ;see if pattern matches
jne      _auto_135          ;if not, try next pattern

```

;we found a framing pattern, set the indication and break out to proceed

```

bset     #6,y:<inpstat        ;indicate the match
enddo
jmp      _auto_145          ;end y:<synccnt loop
                                ;we matched the pattern

```

_auto_135

;try the next framing pattern

```

move     (r0)+

```

_auto_145

;if not a match, we are not framed, try again via framit or autosmpl rtn

```

jclr     #6,y:<inpstat,_auto_155

```

;we did match a framing pattern

```

move     y:<synced,a          ;get count of frames sync'd so far
add      x1,a    y:<syncfrms,x1 ;increment count
                                ; & set to test if limit reached
cmp      x1,a    y:bitoff,r1   ;see if sync frame count reached
                                ; & set the bit offset register

```

SUBSTITUTE SHEET (RULE 26)

-136-

```

        jlt      _auto_75          ;not at limit, go set up for next frame

;we are now considered framed
; indicate OK
; put bit offset for this new frame in proper register
; put address offset for this new frame in proper register
; set the data gathering correctly
; exit the interrupt routine

        clr      a          #>1,x0          ;a=0 indicates we're framed
; & set to set flag to gather data
; r3 is expected to have the bit offset
; address of the last matched frame start
; starting address of input buffer
; calculate the start offset into buffer
; & increment the input write pointer
; save buffer address start word offset
; r5 is expected to have address offset
; set flag for normal data gathering
; done with auto sample this sample rate

        move     y:bitoff,r3
        move     y:wrdoft,b
        move     #syncbuf,x1
        sub      x1,b          ;!!!BEN: (r4)+
;!!!BEN
        move     b,y:wrdoft
;
        move     b,r5
        move     x0,y:<inpstat
        jmp      _auto_160

_auto_155
;failed to frame, indicate to the framit or autosmpl routine to try again

        bset     #8,y:<inpstat

_auto_160
;!!!BEN:perform old ssirec auto sampling on current frame

;
        jset     #0,y:<inpstat,_auto_90 ;framing found
        jset     #8,y:<inpstat,_auto_100 ;conclusion has been as not framed
;
        jmp      _auto_80          ;continue waiting for result

;;_auto_90
;we have successfully framed the correct number of frames in a row
; and therefore we found our sampling rate

;!!!BEN enddo          ;end #NUMSAMPLRATES loop
        bset     #AUTOSAMPLEPROCESS,y:<process ;indicate auto sampling done
        clr      a          ;indicate success to caller
        move     y:linear,m4 ;restore as a linear buffer
        rts       ;return with sample rate found

_auto_100
;!!!BEN
;we did not frame at that last sample rate, try the next one
;turn off the interrupt system
;
        ori      #S03,mr
;
        nop
;
        nop
;
        nop
;
        nop
;
        move     x:srchtries,x0          ;number of tries at sample rate
;
        move     r3,x0          ;number of tries at sample rate

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-137-

```

move    #>MAX_AUTO_TRIES,a    ;get tolerance ctr
cmp     x0,a                    ;see if time to try next sample rate
jgt     _auto_70                ;not yet make another try

;see if the pass looking for frames with privacy bit not set

move    #privacybit,r3          ;addr of privacy bit flag
nop
jset    #0,y:(r3),_auto_108     ;if tried privacy, check protection

;now try looking for a frame header with the privacy bit set

move    #syncptrn,r3            ;modify table of syn patterns
bset    #0,y:privacybit         ;indicate privacy bit set

;for the number sync patterns set the privacy bit set

do      y:<synccnt,_auto_102
bset    #0,y:(r3)+

_auto_102

;restart the attempt counter for the new sync patterns

move    #0,r3
move    r0,x:srchtries          ;zero the failure counter
jmp     _auto_70                ;now make tries with privacy bit set

_auto_108

;see if the pass looking for frames without CRC protection was done
; if so, try next sampling rate

jset    #0,y:<protect,_auto_800 ;if no CRC done, try next sampling rate

;now try looking for a frame header without the CRC protection

move    #syncptrn,r3            ;modify table of syn patterns
bset    #0,y:<protect           ;indicate NO CRC protection
bclr    #0,y:privacybit         ;reset try with privacy bit set to 0

;for the number sync patterns set the NO protection bit

do      y:<synccnt,_auto_110
bset    #8,y:(r3)               ;set the protect bit
bclr    #0,y:(r3)+              ;clear the privacy bit

_auto_110

;restart the attempt counter for the new sync patterns

move    #0,r3
move    r0,x:srchtries          ;zero the failure counter
jmp     _auto_70                ;now make tries without CRC bit

;7/12/94: added label to skip to next sampling rate if not applicable

_auto_800

;this sampling rate did not match, try the next table entry

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-138-

```

;!!!BEN
;;      move    y:<svereg,r0          ;restore sample table address
;;      move    #AUTOBYSAMPLE,n0      ;set auto sample offset to next rate
;;      nop
;;      move    (r0)+n0              ;advance to next sample rate

;!!!BEN: increment the current sample rate table index to try next sample rate

      bclr     #AUTONEXTFRAME,y:<process ;to start next sample rate entry
      move     x:srcrate,b            ;to increment table entry
      move     #>1,x0                 ;increment
      add      x0,b    #>NUMSAMPLERATES,x0 ;increment search index
                                   ; & get max table entries count
      cmp      x0,b    b,x:srcrate     ;see if table totally searched
                                   ; & in case, save new search index
      jlt      _auto_AA               ;if less than max, try new table entry

_auto_900

;we failed to determine the sampling rate, indicate failure to caller

      bset     #AUTOSAMPLEPROCESS,y:<process ;indicate auto sampling done
      move     #>-1,a                 ;indicates failure
      move     y:linear,m4            ;restore as a linear buffer
      rts                                     ;return to caller

```

-139-

```

opt      fc
;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \URDCDSYN\getancda.asm: BEN y:<linear, y:firmemod(inpsize)
;
; This routine decodes the ancillary data bytes for output to rs232 i/f.
;
; on entry
;   r6 = current offset in output array
;   y:dataiptr = address in data byte input buffer to start from
;   y:bytecnt = count of bytes in input buffer not yet transmitted
;
; on exit
;   a = destroyed
;   b = destroyed
;   y0 = destroyed
;   y1 = destroyed
;   r0 = destroyed
;   r1 = destroyed
;   r2 = destroyed
;   r3 = destroyed
;   r4 = destroyed
;   n4 = destroyed
;
;
; include 'def.asm'
; include '..\common\ioequ.asm'
; include 'box_ctl.asm'
;
; section bytebuffer
; xdef  databytes
;
; org      yli:
stgetancda_yli
databytes    ds      DATABUFLEN      ;buffer for bytes received
endgetancda_yli
endsec
;
; section highmisc
; xdef  anctype
; xdef  baudrte
; xdef  dataiptr
; xdef  dataoptr
; xdef  bytecnt
; xdef  maxbytes
; xdef  savea0
; xdef  savea1
; xdef  savea2
; xdef  padbytes
;
; org      yhe:
stgetancda_yhe
anctype      ds      1      ;type of count field after audio data
;           0 = 3 bit padded byte count
;           1 = 8 bit data byte count
baudrte      ds      1      ;data baud rate code from switches
dataiptr     ds      1      ;ptr for next byte decoded from frame

```

SUBSTITUTE SHEET (RULE 26)

-140-

```

dataoptr      ds      1      ;ptr for next byte to transmitted to rs232
bytecnt       ds      1      ;count of bytes yet to be output to rs232
maxbytes      ds      1      ;tolerance check of bytecnt for scixmt
savea0        ds      1      ;save reg a0 for scixmt
savea1        ds      1      ;save reg a1 for scixmt
savea2        ds      1      ;save reg a2 for scixmt
padbytes      ds      1      ;hold pad bytes from the frame

endgetancda_yhe
endsec

      org      phe:

getancdata

;clear the ancillary data problem for old CCS frames
      bclr     #2,y:oldccs

;set address of type of count to extract:
;      padded bits byte count OR data byte count

      move     #ancitype,r4      ;addr of type of count field

;do not decode ancillary data from a reused saved frame
      jset     #USE_SAVED,y:<ctlflgs,_ancd_90 ;if not reused, continue

;see if data byte count, and if so, read byte count and then bytes
      jset     #0,y:(r4),_ancd_78      ;if byte count, get data byte count

;set the end of the MUSICAM portion of the full frame values
      move     y:frendwd,r0      ;normal MUSICAM frame last word address
      move     y:frendbt,n0      ;normal MUSICAM frame last bit offset
      move     y:frmemod,m0      ;set circular buff to addj addr
      move     m0,m1             ;set circular buff to addj addr
      move     #-1,x0            ;init the pad bytes value
      move     x0,y:padbytes

;test if room remaining in the frame to read the CCS ancillary data pad
; byte count
      move     r0,r1             ;get addr of last word into proper reg
      move     r6,a              ;to test next addr to decode
      move     (r1)+             ;to see if last word being decoded
      move     r1,x0             ;to test last frame word address
      cmp      x0,a              ;see if about to decode last
                                ; & set numb bits in pad byte cnt
      jne      _ancd_00          ;if not, test room from curr decode word

;decoding of the last word in the frame is in progress,
; see if sufficient bits remain to get the padded byte count

      move     #>24,b            ;get bits per word
      move     y:<sc,x0           ;get undecoded bits count in last word
      sub      x0,b              ;calc bits decoded from last word so far
                                ; & get total bits in that last word
      neg      b                 ;make bits already decoded negative

```



-141-

```

        add      x0,b          ;add total bits in last word
        cmp      x1,b          ;see if enough bits remain
        jlt      _ancd_85      ;if not it's not CCS, no ancillary data
        jge      _ancd_05      ;if so, do ancillary data
        nop
        nop
        nop
        nop
        jmp      _ancd_85      ;if not it's not CCS, no ancillary data

_ancd_00
;test the next to last word address to test remaining bits - offset to last

        move     (r1)-         ;back up to next to last word addr
        move     r1,x0         ;to test next to last vs next addr
        cmp      x0,a          ;see if next is next to last
        jne      _ancd_05      ;if not at next to last, do ancillary

;see if remaining bits in current (next to last) word being decoded
;plus the number of bits in the last word have enough bits for pad byte cnt

        move     y:<sc,b        ;get undecoded bit cnt curr decode word
        move     n0,x0          ;get total bits in that last word
        add      x0,b          ;add total bits to remaining bits cnt
        cmp      x1,b          ;see if enough bits left in the frame
        jlt      _ancd_85      ;if not, it's not CCS no ancillary data
        jge      _ancd_05      ;if so, do ancillary data
        nop
        nop
        nop
        nop
        jmp      _ancd_85      ;if not, it's not CCS no ancillary data

_ancd_05
;get the count of pad audio bytes from the frame

        move     #masktbl,r2    ;numb bits in pad byte count
        move     #BITSFORPADDING,n4 ;get hi order bit mask index
        move     n4,n2          ;get pad byte count from frame
        jsr      getvalue       ;mask off high order one's
        move     y:(r2+n2),x1    ;mask off high order one's
        and      x1,a          n0,x0 ; & set end of frame bit offset

        move     a1,a           ;clear up for a zero test
        move     a,y:padbytes   ;save the retrieved pad byte count
        tst      a             y:dataiptr,r5 ;test if any pad bytes included
        jeq      _ancd_40      ; & set addr of next byte to be stored
        ;no pad bytes in frame, go decode data

;adjust end of frame for padded bytes (8 bits per byte)

        move     #>8,x1         ;set up bits in a data byte
        move     a1,y1          ;get count of pad bytes
        mpy      x1,y1,a #>24,x1 ;mult by 8 bits per byte
        ; & set bits per word

```


-142-

```

        asr      a      r6,b      ;align integer result:
        move     a0,a      ; & get next decoded word addr
                                ;shift integer result

        _ancd_10
        cmp      x1,a      ;if a full word of padding remains
        jlt      _ancd_20   ;if not, go adjust the bit offset
        move     r0,y0     ;to see if at next decode word
        cmp      y0,b      ;see if next to decode reached
        ;!!!dbg jeq      _ancd_89 ;if so, no data to decode
        jne      _ancd_15   ;if not, keep checking
        nop
        nop
        nop
        nop
        jmp      _ancd_89   ;if so, no data to decode

        _ancd_15
        sub      x1,a      (r0)- ;sub full 24 bits,
                                ; & back off one word in end address
        jmp      _ancd_10   ;try again

        _ancd_20
        ;now back off the number of bits

        cmp      x0,a      x0,b      ;offset vs rest of pad bits
                                ; & offset to b reg for adjustment
        jle      _ancd_30   ;if less or equal, don't adjust
        move     r6,b      ;get next decoded word addr
        cmp      y0,b      x0,b      ;see if next to decode reached
                                ; & offset to b reg for adjustment
        ;!!!dbg jeq      _ancd_89 ;if so, no data to decode
        jne      _ancd_25   ;if not, data to decode
        nop
        nop
        nop
        nop
        jmp      _ancd_89   ;if so, no data to decode

        _ancd_25
        add      x1,b      (r0)- ;adjust offset by bits for full word
                                ; & back off one more word address

        _ancd_30
        ;adjust the bit offset by the remaining pad bits

        move     a,x0      ;get the remaining pad bits
        sub      x0,b      ;calculate new bit offset
        move     b,n0      ;save approx end of anc data offset

        _ancd_40
        ;now get the bytes and store in the buffer for the trasmit interrupt

        move     #DATABUFLEN-1,m5 ;circular buffer
        move     #BITSPERBYTE,n4   ;number of bits to decode from frame
        move     n4,n2             ;get hi order bit mask index

```

BAD ORIGINAL

SUBSTITUTE SHEET (RULE 26)

-143-

```

        move    #0,r3                ;this is the decoded byte counter
_ancd_50:
;as long as there is room for a byte to be decoded, do it

        move    r6,r1                ;curr next frame word address
        move    #>BITSPERBYTE,x1     ;set up bits in a data byte
        move    (r1)-                ;next frame word addr - 1 = curr addr
        move    r0,a                ;get frame end word addr
        move    n0,y0                ;get end bit offset in frame end word
        move    r1,x0                ;to compare curr frame word to end addr
        cmp     x0,a    y:<sc,b       ;is curr frame word equal end frame word
        jne     _ancd_60             ; & get bit offset into curr frame word
                                         ;if not end frame word, try next to last

;since we've decoded into the last word in the frame,
; subtract remaining bit in curr word from 24 to determine how many have
; been decoded
; subtract the used bits from the last word bits available

        move    #>24,a                ;bits per word to be sub from
        sub     b,a    y0,b           ;subtract y:<sc from 24 to get used cnt
                                         ; & get last word bits available
        sub     a,b                    ;sub used bit cnt from bits abvalable
        jmp     _ancd_70             ;see if another byte can be decoded

_ancd_60:
;since we have not reached the last frame word, we must see if we're at
; the next to last frame, and if not, keep decoding ancillary data bytes

        move    r0,r1                ;end frame word address
        nop                                     ;this pains me
        move    (r1)-                ;back up to next to last addr
        move    r1,a                ;for comparison
        cmp     x0,a                ;is curr frame word = end - 1 frame word
        jne     _ancd_75             ;if not, decode the next data byte

;we have reached the next to last frame word,
; add bits from the last frame word to those remaining in this byte
; if there is a byte's worth of bits, decode another ancillary data byte

        add     y0,b                ;add number of bits in last word

_ancd_70:
        cmp     x1,b                ;see if a byte fits in the bits left
        jlt     _ancd_80             ;no more bytes, go update byte count

_ancd_75:
;there is room for another byte, let's get it

        jsr     getvalue              ;retrieve the next byte from the frame
        move    y:(r2+n2),x1         ;mask off high order one's
        and     x1,a    (r3)+        ;mask off high order one's
                                         ; & incr byte counter

;insert the byte into the transmit buffer

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-144-

```

        move    a1,y:(r5)+          ;put the byte out
;test to see that did not exceed baud rate byte count

        move    r3,y0              ;count of data bytes just decoded
        move    y:maxbytes,a       ;maxbytes tolerance decoded check
        cmp     y0,a               ;check for frame alignment error
        jlt     _ancd_85            ;skip if too many bytes decoded
        jmp     _ancd_50            ;see if there is room for another

_ancd_78
;get the count of ancillary data bytes in the frame

        move    #BITSPERBYTE,n4    ;bits in the ancillary data byte count
        move    #masktbl,r2        ;set addr of the masking table
        move    n4,n2              ;get hi order bit mask index
        jsr     getvalue            ;get pad byte count from frame
        move    y:(r2+n2),x1        ;get mask off high order one's
        and     x1,a #0,r3         ;mask off high order one's
        ; & zero decoded byte counter
        move    a1,a               ;clean up for a zero test
        tst     a                  ;test if any data bytes included
        ; & set addr of next byte to be stored
        jeq     _ancd_90            ;no data bytes in frame, we're done

;make sure the data byte count is valid vs the max bytes at this baud rate

        move    y:maxbytes,x0      ;get max bytes @ baud rate
        cmp     x0,a               ;comp byte count from frame to max
        jgt     _ancd_85            ;if number is too big, skip data

;now get the bytes and store in the buffer for the transmit interrupt

        move    #DATABUFLEN-1,m5   ;set circular buffer

;get the count of ancillary data bytes in the frame
; bytes are stored in the reverse order received by encoder

        do      a,_ancd_80

;get the next ancillary data byte

        jsr     getvalue            ;retrieve the next byte from the frame
        move    y:(r2+n2),x1        ;mask off high order one's
        and     x1,a (r3)+          ;mask off high order one's
        ; & incr byte counter

;insert the byte into the transmit buffer

        move    a1,y:(r5)+          ;put the byte out

_ancd_80
;temporarily disable the interrupt for data received

        bclr    #M_TIE,x:<<M_SCR
        nop
        nop

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-145-

```

        nop

;while waiting for interrupt to take effect:
;   make a tolerance check of the frame's alignment to make sure
;   we haven't decoded more data bytes than is possible
;   if we have decoded too many bytes,
;   skip the junk just decoded by ignoring the results of this frame

        move    r3,y0                ;count of data bytes just decoded
        move    y:maxbytes,a         ;maxbytes tolerance decoded check
        cmp     y0,a    y:bytecnt,a  ;check for frame alignment error
                                        ; & get latest byte cnt of unsent bytes
        jlt     _ancd_85             ;skip if too many bytes decoded

;interrupt should now be disabled and we can safely update count of unsent bytes

        add     y0,a    r5,y:dataiptr ;add count of bytes just framed
                                        ; & save addr of next byte next frame
        move    a,y:bytecnt          ;save new unsent byte count
        jmp     _ancd_89             ;reset interrupt

_ancd_85

;a problem decoding ancillary data may indicate a stream of frames from
;   some other manufacturer
;   or,
;   if the frames are from a CCS encoder that is encoding old CCS CDQ2000
;   two-channel frames at a low bit rate that is incorrectly using
;   the wrong allowed table BUT, has an old CCS CRC-16 checksum

;!!!dbg
        nop
        nop
        nop
        nop
        nop
;!!!dbg
        jset    #CRC_OLD_vs_NEW,y:<ctifigs,_ancd_89    ;if ISO CRC, continue
;!!!dbg
        nop
        nop
        nop
        nop
        nop
;!!!dbg
        bset    #2,y:oldccs                ;show problem to switch to old CCS

_ancd_89

;turn the transmit byte interrupt back on

        bset    #M_TIE,x:<<M_SCR            ;enable transmit interrupt

;return after all bytes decoded and counted

        move    y:linear,m0                ;uncircular buffer
        move    m0,m1                      ;uncircular buffer
        move    m0,m5                      ;uncircular buffer

_ancd_90

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



WO 96/32805

PCT/US96/04835

rcs

-146-

SUBSTITUTE SHEET (RULE 26)

-147-

```

opt      fc

(c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
\URDCDSYN\getbal.asm: BEN y:<frmttype y:<sibound

title    'Get bit allocations'

This routine is used to get the bit allocations of each of the sub-bands.
It is from the ISO standard.
sub-band 0 - 10 use 4 bits (11 * 4 = 44 bits)
sub-band 11 - 22 use 3 bits (12 * 3 = 36 bits)
sub-band 23 - 26 use 2 bits ( 4 * 2 =  8 bits)
                ( total = 88 bits)

on entry
    r0 = address of bit allocation array for both left and right channels
    r6 = current offset in the input array
    n6 = base address of the input array
    y:<maxsubs = MAXSUBBANDS at sampling rate and bit rate
    y:sc = shift count of current input word
    y:frmttype = full stereo, joint stereo or mono
    y:sibound = joint stereo sub-band intensity bound
    x:crclbits = accumulator of bits covered by CRC-16 routine
                (bit allocation bits are accumulated)

on exit
    r6 = updated
    y:sc = updated

    a = destroyed
    b = destroyed
    x0 = destroyed
    x1 = destroyed
    y0 = destroyed
    y1 = destroyed
    r0 = destroyed
    r1 = destroyed
    r2 = destroyed
    r4 = destroyed
    n4 = destroyed

include 'def.asm'

section highmisc
xdef    masktbl
xdef    tbl

org      yhe:
stgetbal_yhe:

masktbl
dc       $000000    ;placeholder in mask table
dc       $000001    ;mask table for 1 bit getvalue
dc       $000003    ;mask table for 2 bit getvalue
dc       $000007    ;mask table for 3 bit getvalue
dc       $00000f    ;mask table for 4 bit getvalue
dc       $00001f    ;mask table for 5 bit getvalue

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-148-

```

dc      $00003f      ;mask table for 6 bit getvalue
dc      $00007f      ;mask table for 7 bit getvalue
dc      $0000ff      ;mask table for 8 bit getvalue
dc      $0001ff      ;mask table for 9 bit getvalue
dc      $0003ff      ;mask table for 10 bit getvalue
dc      $0007ff      ;mask table for 11 bit getvalue
dc      $000fff      ;mask table for 12 bit getvalue
dc      $001fff      ;mask table for 13 bit getvalue
dc      $003fff      ;mask table for 14 bit getvalue
dc      $007fff      ;mask table for 15 bit getvalue
dc      $00ffff      ;mask table for 16 bit getvalue
dc      $01ffff      ;mask table for 17 bit getvalue
dc      $03ffff      ;mask table for 18 bit getvalue
dc      $07ffff      ;mask table for 19 bit getvalue
dc      $0ffffff      ;mask table for 20 bit getvalue
dc      $1ffffff      ;mask table for 21 bit getvalue
dc      $3ffffff      ;mask table for 22 bit getvalue
dc      $7ffffff      ;mask table for 23 bit getvalue
dc      $fffffff      ;mask table for 24 bit getvalue

;define data size table for the getvalue routine to extract data
tbl
dc      $000000      ;bits = 0, place holder
dc      $000001      ;shift left 01 bits
dc      $000002      ;shift left 02 bits
dc      $000004      ;shift left 03 bits
dc      $000008      ;shift left 04 bits
dc      $000010      ;shift left 05 bits
dc      $000020      ;shift left 06 bits
dc      $000040      ;shift left 07 bits
dc      $000080      ;shift left 08 bits
dc      $000100      ;shift left 09 bits
dc      $000200      ;shift left 10 bits
dc      $000400      ;shift left 11 bits
dc      $000800      ;shift left 12 bits
dc      $001000      ;shift left 13 bits
dc      $002000      ;shift left 14 bits
dc      $004000      ;shift left 15 bits
dc      $008000      ;shift left 16 bits
dc      $010000      ;shift left 17 bits
dc      $020000      ;shift left 18 bits
dc      $040000      ;shift left 19 bits
dc      $080000      ;shift left 20 bits
dc      $100000      ;shift left 21 bits
dc      $200000      ;shift left 22 bits
dc      $400000      ;shift left 23 bits
dc      $800000      ;shift left 24 bits

endgetbal_yhe
endsec

section highmisc
xdef    skftbl
xdef    skftbl_1
xdef    skftbl_2
xdef    skftbl_3

org      xhe:
stgetbal_xhe

```

SUBSTITUTE SHEET (RULE 26)

-149-

;address of BAL's bit table as per Allowed table selected

skftbl ds 1

;These tables is the number of bits used by the scale factor in each sub-band

; High sampling rates with higher bit rate framing

skftbl_1

dc	4	;sub-band 0
dc	4	;sub-band 1
dc	4	;sub-band 2
dc	4	;sub-band 3
dc	4	;sub-band 4
dc	4	;sub-band 5
dc	4	;sub-band 6
dc	4	;sub-band 7
dc	4	;sub-band 8
dc	4	;sub-band 9
dc	4	;sub-band 10

dc	3	;sub-band 11
dc	3	;sub-band 12
dc	3	;sub-band 13
dc	3	;sub-band 14
dc	3	;sub-band 15
dc	3	;sub-band 16
dc	3	;sub-band 17
dc	3	;sub-band 18
dc	3	;sub-band 19
dc	3	;sub-band 20
dc	3	;sub-band 21
dc	3	;sub-band 22

dc	2	;sub-band 23
dc	2	;sub-band 24
dc	2	;sub-band 25
dc	2	;sub-band 26

;end table 3-B.2a

dc	2	;sub-band 27
dc	2	;sub-band 28
dc	2	;sub-band 29

;end table 3-B.2b

dc	2	;sub-band 30
dc	2	;sub-band 31

; High sampling rates with lower bit rate framing

skftbl_2

dc	4	;sub-band 0
dc	4	;sub-band 1
dc	3	;sub-band 2
dc	3	;sub-band 3
dc	3	;sub-band 4
dc	3	;sub-band 5
dc	3	;sub-band 6
dc	3	;sub-band 7

SUBSTITUTE SHEET (RULE 26)

-150-

```

;end table 3-B.2c
dc 3 ;sub-band 8
dc 3 ;sub-band 9
dc 3 ;sub-band 10
dc 3 ;sub-band 11
;end table 3-B.2d
dc 3 ;sub-band 12
dc 3 ;sub-band 13
dc 3 ;sub-band 14
dc 3 ;sub-band 15
dc 3 ;sub-band 16
dc 3 ;sub-band 17
dc 3 ;sub-band 18
dc 3 ;sub-band 19
dc 3 ;sub-band 20
dc 3 ;sub-band 21
dc 3 ;sub-band 22
dc 3 ;sub-band 23
dc 3 ;sub-band 24
dc 3 ;sub-band 25
dc 3 ;sub-band 26
dc 3 ;sub-band 27
dc 3 ;sub-band 28
dc 3 ;sub-band 29
dc 3 ;sub-band 30
dc 3 ;sub-band 31

```

; Low sampling rates

```

skfcb1_3
dc 4 ;sub-band 0
dc 4 ;sub-band 1
dc 4 ;sub-band 2
dc 4 ;sub-band 3

dc 3 ;sub-band 4
dc 3 ;sub-band 5
dc 3 ;sub-band 6
dc 3 ;sub-band 7
dc 3 ;sub-band 8
dc 3 ;sub-band 9
dc 3 ;sub-band 10

dc 2 ;sub-band 11
dc 2 ;sub-band 12
dc 2 ;sub-band 13
dc 2 ;sub-band 14
dc 2 ;sub-band 15
dc 2 ;sub-band 16
dc 2 ;sub-band 17
dc 2 ;sub-band 18
dc 2 ;sub-band 19
dc 2 ;sub-band 20
dc 2 ;sub-band 21
dc 2 ;sub-band 22
dc 2 ;sub-band 23
dc 2 ;sub-band 24
dc 2 ;sub-band 25
dc 2 ;sub-band 26
dc 2 ;sub-band 27

```

SUBSTITUTE SHEET (RULE 26)

-151-

```

        dc      2      ;sub-band 28
        dc      2      ;sub-band 29
;end table 3-B.1
        dc      2      ;sub-band 30
        dc      2      ;sub-band 31

endgetbal_xhe
endsec

org phe:

;initialize:
; a. r1 with start of subband allocation table of bits in frame per sub-band
; b. n0 offset for right channel sub-band bit allocation values:
;    left channel from 0 to (NUMSUBBANDS - 1)
;    right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)
; c. r3 set with joint stereo sub-band boundary for stereo intensity:
;    4 (4-31), 8 (8-31), 12 (12-31) or 16 (16-31)
;
getbal:
        move     x:skftbl,r1
        move     #masktbl,r2
        move     #NUMSUBBANDS,n0      ;offset for right channel
        move     y:sibound,r3        ;decr stereo intens sub-band ctr
        move     x:crchits,r5        ;get CRC-16 bit counter

;loop through the sub-bands extracting the left and right (if applicable)
;bit allocation index values (y:<maxsubs = fixed count of sub-bands framed):
; a. for current sub-band get the number of bits for allocation index value
;    and increment address of the next sub-band bit count
; b. get the bit allocation for the left channel always
; c. b register isolate the type of frame: full stereo, joint stereo or mono
; d. y0 holds the mono frame type code for testing
; e. y1 holds the joint stereo frame type code for testing
; f. see if the frame type is joint stereo and just in case, move the
;    current stereo intensity sub-band boundary counter value for testing
; g. if not joint stereo, see if this is a mono frame type
; h. if it is joint stereo:
;    1. test if the boundary counter has reached zero, and just in case it has,
;       restore the left channel bit allocation value to the a1 register
;    2. if the counter is zero, go to copy left channel into the right channel
;    3. if not, go to extract the full stereo right channel allocation value

        do       y:<maxsubs,_getb_40
        move     x:(r1)+,n4           ;get # of bits to read
        move     n4,n2               ;get hi order bit mask index
        move     n4,n5               ;to accumulate CRC-16 bits
        jsr      getvalue            ;get a left chan bit allocation
        move     y:(r2+n2),x1        ;mask for high order one's
        move     (r5)+,n5            ;accum bits for CRC-16 rtn
        and      x1,a      y:frmtype,b ;mask off high order one's
                                           ; & set for frame type compare
        move     a1,x:(r0)           ;set left channel
        move     #>MONO,y0          ;ck for no right channel
        move     #>JOINT_STEREO,y1  ;ck for intensity sub-band
        cmp      y1,b      r3,a     ;check for stereo intensity
        jne      _getb_10           ;if not, see if mono
        tst      a      x:(r0),a1    ;reached bound, restore left val
        jeq      _getb_30           ;yes, left val to right val
        move     (r3)-              ;no, decr intens sub-band ctr

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-152-

```

        jmp      _getb_20      ; and retrieve right chan value
; test for a mono type of frame and just in case it is, set a1 to zero
; for insertion into the right channel for consistency
; if it is mono, go to move the right channel value
; otherwise, fall through to full stereo
_getb_10
        cmp     y0,b    #0,a1      ; if mono, insert 0 for right
        jeq     _getb_30
; full stereo, extract the right channel bit allocation value
_getb_20
        jsr     getvalue      ; get a right chan bit allocation
        move    y:(r2+n2),x1    ; mask for high order one's
        move    (r5)+n5        ; accum bits for CRC-16 rtn
        and     x1,a          ; mask off high order one's
; insert the right channel value (n0 offset)
; increment for the next sub-band
_getb_30
        move    a1,x:(r0+n0)      ; right channel sub-band alloc
        move    (r0)+            ; incr for next sub-band
_getb_40
; Fill the unused sub-bands with 0 bit allocation
; This allows getdata to process these sub-bands normally and insert 0
; data in them.
        clr     a    #>NUMSUBBANDS,b
        move    y:<maxsubs,x0    ; current MAXSUBBANDS
        sub     x0,b            ; equals unused sub-bands
        dc     b,_getb_50
        move    a,x:(r0+n0)      ; right channel
        move    a,x:(r0)+        ; left chan & incr for next
_getb_50
        move    r5,x:crcbits      ; store updated CRC-16 bit counter
        rts

```

-153-

opt: fo:mex

(c) 1995. Copyright Corporate Computer Systems, Inc. All rights reserved.

\DGCST\rmicrmus.asm: with Reed Solomon decoding

title 'Main'

27/4/93: rmicrmus.asm version of cdq2000 MUSICAM (rdcdsynt.asm) for micro

08/26/91: (dsb & lwh)

NOTE: Never use m4 to control a circular buffer. The interrupt routine, ssirec.asm has been sped up by using m4 and then restoring it to a linear buffer.

; This routine does it all for the decoder.

include 'def.asm'

include '..\common\ioequ.asm'

include 'box_ctl.asm'

section highmisc

xdef SBndSKF

xdef ASMDData ;set A of 192 inverse quantized l&r

org xhe:

strmicro_xhe

SBndSKF ds NUMSUBBANDS*NPARGROUP*2 ;left & right sub-band scale factors
 ASMDData ds NUMSUBBANDS*NPARGROUP*2 ;192 samples per 1 group of 3 samples
 ; for 32 sub-bands from both channels

endrmicro_xhe

endsec

section highmisc

xdef checksum

xdef frm_size

xdef frmmod

xdef frmhalf

xdef framesz

xdef oof

xdef voof

xdef poof

xdef doof

xdef IPwrdoff

xdef IPbitoff

xdef wrdoff

xdef bitoff

xdef dcdfrmod

xdef sveidbit

xdef sverate

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-154-

```

xdef svesmpl
xdef smplcde
xdef bitrate
xdef inpaddr
xdef frmrate
xdef smplrte
xdef iputcde
xdef smplidbit
xdef maxsubs_1
xdef maxsubs_2
xdef oldccs
xdef biterrs
xdef fade
xdef fadeent
xdef frtries
xdef sampling,bitrates,baudcik

org yhe:
strmicro_yhe

checksum ds 1 ;hold checksum from coded frame
frmsize ds 1 ;number of words in a frame
frmmod ds 1 ;numb words in 2 frames - 1 (mod buffer)
frnhalf ds 1 ;1/2 words in framed buf (rd ptr check)
framesz ds 1 ;size of framing input mod buffer ctl

;successive framing faults:
; oof = out-of-frame sync pattern failures
; vocf = sample rate code faults (auto sample vs frame header)
; poof = CRC protection code faults (auto sample vs frame header)
; doof = ancillary data errors coupled with old CCS CRC-16 algorithm
oof ds 1 ;out-of-frame faults: numb of oof's (0-NOOF)
vocf ds 1 ;number of vocf's (0-NOOF)
poof ds 1 ;CRC protection faults: numb of poof's (0-NOOF)
doof ds 1 ;ancil data with old CCS CRC-16: doof's (0-NOOF)
IPwrdoff ds 1 ;frame 1/p word offset from start of buffer
IPbitoff ds 1 ;frame 1/p bit offset from msb
wrdoff dc 0 ;frame decoding word offset from start of buffer
bitoff dc 0 ;frame decoding bit offset from msb
dcdfrmod ds 1 ;framebuf circ buf mod ctl

;these are for auto detect as requested by switches

sveidbit ds 1 ;ISO sampling id bit from frame header: low/high
sverate ds 1 ;ISO bit rate from frame header: lo/hi Kbit rate
svesmpl ds 1 ;ISO sampling rate from frame header: low/high
smplcde ds 1 ;ISO sampling rate from on select sws: low/high
bitrate ds 1 ;ISO bit rate from select sws: lo/hi Kbit rate

inpaddr ds 1 ;hold i/p buf addr to restore after save
frmrate dc 0 ;dip switch (1 bit) indicate which
; of 2 selectable bit rates
; bit rate sets numb words in a frame:
; 0 = lower Kbit rate
; 1 = higher Kbit rate

smplrte dc 0 ;i/p PCM data sampling rate
iputcde dc 0 ;0 = MUSCIMAM frames, 1 = G722 data i/p
smplidbit dc 0 ;ISO hdr id bit:
; 1 = 32 or 48 K sampling rate
; 0 = 16 or 24 K sampling rate
maxsubs_1 ds 1 ;MAXSUBBANDS is MONO frames

```

SUBSTITUTE SHEET (RULE 26)

[BAD ORIGINAL]

-155-

```

maxsubs_2    ds    1    ;MAXSUBBANDS if 2 channel frames
oldccs       ds    1    ;bit 0 = 1 to decode old CCS CDQ1000
                    ; 0 means MPEG-ISO frames
biterrs ds    1    ;count successive bit errors
fade ds    1    ;in case of fade volume output ccl
fadecnt ds    1    ;in case of fade volume output ccl
frtries dc    0    ;count framing to reboot if too many

```

```

SAMPLERATES    ;table of sample rate variables

```

```

BITRATES       ;table of framing bit rate variables

```

```

BAUDCLK        ;table of specified ancillary data rates

```

```

endrmicro_yhe
endsec

```

```

;The variables below are defined in lowmisc in low y memory and must be located
; below address 40 to make use of short addressing.

```

```

section lowmisc
xdef word_out,word_in,not_appl
xdef frmtype
xdef sibound
xdef cclflgs
xdef maxsubs
xdef protect
xdef inpstat
xdef inpsize
xdef temp
xdef olwptr,orwptr
xdef linear

```

```

org yli:
strmicro_yli

```

```

word_out ds    1    ;applicable hardware outputs (leds, switches)
word_in ds    1    ;applicable hardware inputs (switches, lines)
not_appl ds    1    ;satisfy non-applicable hardware settings
frmtype ds    1    ;from coded frame indicates:
                    ; 00 = (0) full stereo
                    ; 01 = (1) joint stereo
                    ; 10 = (2) dual channel
                    ; 11 = (3) mono (1 channel)
sibound ds    1    ;intensity subband boundary alloc addr
cclflgs ds    1    ;control indicators in certain bits:
                    ; bit 0 = STEREO_vs_MONO:
                    ; 0 = stereo
                    ; 1 = mono
                    ; bit 2 = joint stereo or not
                    ; 0 = NOT joint
                    ; 1 = joint stereo frame
                    ; bits 6, 7 and 8 indicate protection
                    ; was a saved frame used 0=no, 1=yes
                    ; bit 6 is overwritten when validating
                    ; the checksum after getsbits:
                    ; if 0 = checksum valid,
                    ; use the frame in progress

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-156-

```

; and save it when finished
; if 1 = checksum failed,
; use previous saved frame
; and bypass saving it when done
; bit 7 indicates if a saved frame
; has been stored:
; 0 = no saved frame
; 1 = yes a saved frame
; bit 8 indicates to getvalue this
; is a good frame to store:
; 0 = do not store in save area
; 1 = do store in save area
; bit 18 indicates whether the frame
; is coded with CRC protection or not:
; 0 = no CRC16 checksum
; 1 = yes CRC16 checksum included
; bit 19 is for mono output only when
; one channel is used for output and
; the other is to be muted
; (see bit 20):
; 0 = left channel for output
; 1 = right channel for output
; bit 20 is for mono output only and
; specifies if the mono is to output
; to one or both channels:
; 0 = both channels
; 1 = one channel only
; as defined by bit 19
; working MAXSUBBANDS
; flag for CRC checksum protection:
; bit 0: 0 = yes, 1 = no
; state of data collection
; used by ssirec to set mod buffer i/p
; use by ssixmte for temp storage
; output left write pointer
; output right write pointer
; value -1 to reset regs to linear buffs

maxsubs ds 1
protect ds 1

inpstat ds 1
inpsize ds 1
temp ds 1
olwptr ds 1
orwptr ds 1
linear ds 1

endrmicro_yli
endsec

org phe:

start

; turn off the interrupt system

ori #S03, mr
nop
nop
nop

movep #S0001.x: <<M_BCR ; set all external io wait states

; set dsp56002 clock to selected MHz (PLL Control Register)

RDECODE_M_PCTL

; jsr <initdeb ; init the debug port
; move #S720906, a

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-157-

```

        jsr     <outhex
        jsr     <cr

; initialize the volume output fade control

;FD      clr     a
;FD      move    a,y:fade
;FD      move    a,y:fadecnt

; PORT C Assignments
;
; s = ssi port
; i = input port
; o = output port
;
; B - 7 6 5 4 - 3 2 1 0
; s s s s s i s s

RDECODE_PORT_C_M_PCC      ;set C control register for general IC
RDECODE_PORT_C_M_PCD      ;set the default outputs
RDECODE_PORT_C_M_PCDDR     ;set C register direction

; initialize the ssi port for the input from the xmitter

RDECODE_SSI_M_CRA          ;set ssi cra register
RDECODE_SSI_M_CRB          ;set ssi crb register

; initialize the sci port for tty

RDECODE_SCI_M_SCR          ;set sci status control register

; PORT B Assignments
;
; i = input port
; o = output port
;
; 14 13 12 - 11 10 9 8 - 7 6 5 4 - 3 2 1 0
; o o o o o o o o o i i i i i i i i

RDECODE_PORT_B_M_PBC      ;set B control register for general IC
RDECODE_PORT_B_M_PBD      ;set the default outputs
RDECODE_PORT_B_M_PBDDR     ;set B register direction

move     #>ON_LEDS_DCD,b      ;flash the LEDs on
move     b,y:<word_out
CLR_DAC_RESET              ;clear the DAC reset line to mute output
ON_LO_SAMPLE_RATE_LED_DCD
ON_HI_SAMPLE_RATE_LED_DCD
SET_LEDS_DCD
INTERRUPT_HOST_DCD
move     #>RDCDSYNT_STARTUP,a
jsr      <wait

; initialize the linear buffer value for mX

move     #-1,m0              ;reset to a linear buffer
move     m0,y:<linear

; init the auto select test table of frame lengths, sample rate and bit rate
; this table as each entry with 2 words: length, sample/bit flags

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-158-

```

; bit : of flag word indicates sample rate: 0 = low, 1 = high
; bit : of flag word indicates framing bit rate: 0 = low, 1 = high

move    #autotbl,r0          ;table of selectable frame lengths
move    #testtbl,r1         ;table to test from
move    x:(r0)+,x0          ;get 1st entry frame length
move    x0,x:(r1)+          ;store smallest frame
move    #>1,x0              ;indicate high sample/low bit rates
move    x0,x:(r1)+
move    x:(r0)+,x0          ;2nd smallest frame
move    x0,x:(r1)+          ;indicate high sample/high bit rates
move    #>3,x0
move    x0,x:(r1)+
move    x:(r0)+,x0          ;2nd largest frame
move    x0,x:(r1)+          ;indicate low sample/low bit rates
move    #0,x0
move    x0,x:(r1)+
move    x:(r0)+,x0          ;largest frame
move    x0,x:(r1)+          ;indicate low sample/high bit rates
move    #>2,x0
move    x0,x:(r1)

;set start-up auto selects

bset    #0,x:autorate        ;with lower bit rate
bset    #0,x:autocode        ;as MUSICAM
bset    #0,x:autosmpl        ;at low sample rate 24.000

restart

CLR_DAC_RESET                ;clear the DAC reset line to mute output
INTERRUPT_HOST_DCD

;turn off the interrupt system
; set the interrupt for host interrupts
; HOST set to IPL 2

movep   #>S0800,x:<<M_IPR    ;set int priorities and edges
andi    #Sfc,mr              ;turn on the interrupt system

ori     #S03,mr

;disable the ancillary data transmit interrupt

bclr    #M_TIE,x:<<M_SCR

; bit 0
; The input state word, y:inpstat, controls data collection from the outside
; into the decoder. If bit 0 is 0, then everytime an input occurs, event is
; counted by incrementing the input write pointer (y:inpwptr) and no data is
; stored. If bit 0 is a 1, then data is stored and the input write pointer
; is incremented.

clr     a                    ;initialize leds as off
move    a,y:<inpstat         ;state of the input buffer
move    a,y:<ctlfigs         ;decoding control flags
move    a,y:<nct_app         ;clear any stubbed flags

;initialize the led output word and light initial leds

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-159-

```

move    d,y:<word_out
ON_ALARM_LED_DCD          ;light alarm led indicator
TST_SET_ALARM_RELAY_DCD,_set_led_0 ;unless already set.
SET_ALARM_RELAY_DCD       ;set the alarm relay line on

```

```

_set_led_0
OFF_LO_SAMPLE_RATE_LED_DCD
OFF_HI_SAMPLE_RATE_LED_DCD

```

```

.....
TEST NOTICE THAT THE FOLLOWING DATA IS DECODED AND PUT INTO A HIGH MEMORY
AND WILL BE CHECKED WITH THE CODED DATA ALL THE TIME WHILE THE PROGRAM
RUNS TO MAKE SURE THAT NONE OF A WORD IS IN ERROR
TEST DATA

```

```

;initialize the buffer to be decoded for testing

```

```

OFF_REED_SOL_LED_DCD          ;indicate no problem with Reed Solomon

move    y:<linear,m1          ;make sure it's linear buffer
move    y:<linear,m3          ;make sure it's linear buffer
move    y:<linear,m6          ;make sure it's linear buffer

move    #framebuf,r1         ;code the 1st of the encoded frames
clr     a                    ;zero the test value accumulator
        #>1,x0               ; & to increment in the test buffer

```

```

;set the frame buffer to sequentially incremented values

```

```

do      #96,_init1
add     x0,a
mcve    a1,x:(r1)-

```

```

_init1

```

```

;do the reed solomon encoding on the test frame buffer

```

```

move    #syncbuf,r1          ;o/p pointer of buffer to be RS-DECODED
move    #RStest,r6           ;i/p pointer for CODED data to decode
move    #PROF1,r3            ;Reed Solomon profile: control decode
jsr     <rsdec16             ;encode via reed solomon

```

```

;test if the reed solomon codec worked or NOT

```

```

move    #syncbuf,r6          ;pointer for DECODED data to be stored
move    #framebuf,r1         ;pointer for the verification table

```

```

;verify that the reed solomon coded values are correct

```

```

do      #86,_RS_Chk          ;Get current coded data output
move    x:(r6)+,x0           ;Get precoded look up table value
move    x:(r1)+,a            ;compare 2 values
cmp     x0,a                 ;if SAME No problem
jeq     <_Same               ;indicate no problem with Reed Solomon
ON_REED_SOL_LED_DCD
enddc
nop

```

```

_Same
nop

```



-160-

_RS_Chk

```

        SET_LEDS_DCD
        INTERRUPT_HOST_DCD

;mute current output buffer

        move    #outbuf,r7          ;setup synth variables
        jsr     <muteout             ;mute the dac output buffer

;get the external switches to determine frame bit rate
; and ancillary data baud rate

        GET_SWITCHES_DCD gsws_00
        jsr     <getsws

;MUSICAM selections by switches set up prior to possible auto select

        move    x:tstsmpl,y1
        move    y1,y:smplrte        ;set the i/p PCM sampling rate code
        move    x:tstcode,y1
        move    y1,y:iputcode       ;set type of i/p data MUSICAM vs G722
        move    x:tstrate,y1
        move    y1,y:frmrate        ;set the frame rate i/p code

;!!!dsb 11/22/94
;::if no auto selection required, go with the settings from the input switches
;::
;::    move    #autosel,r0
;::    nop
;::    jclr    #0,x:(r0),_onward_    ;NO auto selection required
;::
;!!!dsb 11/22/94

;if the selection of MUSICAM vs G722 is not auto selected,
; test for MUSICAM input data stream selected versus G722 data input stream
; and if G722 selected manually, boot rom file from lower half of the chip

        jset    #AUTO_SELECT_DATA_TYPE,y:<ctrlflgs,_auto_type
        move    y:iputcode,b
        tst     b                    ;0 = MUSICAM, else G722
        jne     <g722_boot           ;if 1, it's G722, boot lower half

_auto_type

;initialize the auto select MUSICAM max tries

        move    #>MAX_BOOT_TRIES,x0
        move    x0,x:maxtries
        jsr     <autoselect          ;try for MUSICAM input data

;if autoselect successful, use the selected info

        move    #autosel,r0
        nop
        nop
        nop
        jclr    #0,x:(r0),_onward_    ;NO auto selection required

;if auto select for MUSICAM_vs_G722, it must be G722

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-161-

```

nop
nop
nop
nop
nop
jset    #AUTO_SELECT_DATA_TYPE,y:<ctrlf:gs.g722_boc:

;indicate not MUSICAM framed

ON_FRAME_LED_DCD          ;set the framing led alarm
SET_LEDS_DCD
INTERRUPT_HOST_DCD
jmp     <restart          ;try for new switch settings

_onward_

;everything for MUSICAM selected by switches or auto selection

move    x:tstsmpl,y:      ;set the i/p PCM sampling rate code
move    y1,y:smplrte
move    x:tstcode,y1
move    y1,y:iputcode     ;set type of i/p data MUSICAM vs G722
move    x:tstrate,y1
move    y1,y:frmrate      ;set the frame rate i/p code
move    x:tstbaud,y1
move    y1,y:baudrate     ;set ancillary data baud rate code

;test for the diagnostic method of operation

TST_CLR_DIAGNOSTICS_DCD,_go_fwd ;if normal operation, continue

;diagnostic method of operation selected, reboot from the low portion of chip

bclr    #1,x:<M_PBD        ;clr boot c000 for rdcdiag boot c000:

jmp     <bootup

_go_fwd

; set the values for the data collection routine.
; This is used for setting the value for the mod buffer ctrl
; y:framesz      input for purposes of framing
; y:frmmod       normal framed input (double buffered-2 frames)
; but setting the address of a buffer (y:inpwptr) can't hurt either.

move    #syncbuf,a0        ;set input word pointer
move    a0,y:<inpwptr
move    #framebuf,a0       ;buffer addr of MUSICAM decode buffer
move    a0,y:inpaddr       ;store input buf addr for saving frame

;set access to the flags resulting from autoselect framing pattern match:
; bit 0 - sampling rate: 0 = low, 1 = high
; bit 1 - framing bit rate: 0 = low, 1 = high
; bit 2 - ISO vs old CCS: 0 = ISO, 1 = old CCS CDQ1000
; bit 3 - CRC-16 protection: 0 = yes, 1 = unprotected

move    #chkflags,r1       ;to test results of autoselect match

;based on the sampling rate and framing bit rate selected:

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-162-

```

set the sampling rate code for the ISO frame header
set the framing bit rate code for the ISO frame header
set the frame size in words and bits

move    #sampling,r0          ;addr of sampling rate codes
move    y:smplrte,b           ;offset to sampling code table
tst     b                     ;test for sampling rate of zero
jeq     <_smplcds_            ; & set register to advance thru table
                                ;if code is zero, we're there

rep     b
move    (r0)+n0               ;position to selected sampling rate code

_smplcds_
move    #4,n0                 ;offset MPEG-ISO vs old CCS values
jclr    #2,x:(r1),_smpl_cds_  ;if ISO, r0 is all set for ISO values
move    (r0)+n0               ;offset to old CCS CDQ1000 values

_smpl_cds_
move    y:(r0)+,x0            ;get frame header sampling code
move    x0,y:smplcde          ;save code to match in the frame header
move    y:(r0)+,x0            ;get frame header sampling id bit
move    x0,y:smplidbit        ;save code to match in the frame header
move    y:(r0)+,x0            ;get 1 channel frame maximum sub-bands
move    x0,y:maxsubs_1        ;save max sub-bands for decoding mono
move    y:(r0)+,x0            ;get 2 channel frame maximum sub-bands
move    x0,y:maxsubs_2        ;save max sub-bands for decoding dual
move    y:frmrate,b           ;test bit rate to set audio data size
move    #bitrates,r0          ;addr of framing bit rate info
tst     b                     ;test for rate of zero
jeq     <_bit_offs_          ; & set register to advance thru table
                                ;if code is zero, we're there

rep     b
move    (r0)+n0               ;position to selected bit rate code

_bit_offs_
;set the table offset based on sampling rate

move    y:smplrte,b           ;get the sample rate code
tst     b                     ;test if low sampling rate
jeq     _bit_smpl_            ; & set offset to proper sampling rate
                                ;if low rate, addr is set

rep     b
move    (r0)+n0               ;position to selected sample rate

_bit_smpl_
move    y:(r0)+,x0            ;get ISO bit rate code in frame header
jclr    #2,x:(r1),_bit_rate_  ;if ISO, x0 is all set with ISO code
move    y:(r0),x0             ;get old CCS bit rate code in frame hdr

_bit_rate_
move    x0,y:bitrate          ;save frame header bit rate code
move    #>1,x0                ;to subtract 1 for mod buffer ctl below
move    (r0)+                 ;advance to sampling rate lengths
move    y:(r0),b              ;kbit/sec rate frame size in words
move    b,y:frmsize           ;set # of words in a frame
sub     x0,b                  ;to set decode framebuf mod ctl

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-163-

```

move    b1,y:d2dfmod      ;set MUSICAM decode framebuf mod ctrl
move    y:frmsize,b        ;get # of words in a frame
lsl     b                  ;double buffer framed i/p buffer
sub     x0,b    #>NSBUFS,x1 ;subtract 1 for mod buffer control
                        ;& set number of frames to check
move    b1,y:frmmod        ;save mod buffer control - 2 frames
add     x0,b    y:frmsize,y1 ;re-add 1 to calculate 1/2 frame size
                        ; and get full frame for below
lsl     b                  ;frame size divided by 2
move    b1,y:frmhalf       ;save 1/2 frame size (1 full frame)

;now calculate the framing buffer circular mod control size

mpy     x1,y1,a #>1,y0     ;times frame size
                        ; and set up 1 to decrement
asr     a                  ;align integer result
move    a0,a              ;shift integer result
sub     y0,a              ;minus 1 for mod buffer control
move    a1,y:framesz       ;save framing mod buffer control

;set up for ancillary data to be decoded from a framed and transmit via rs232
; a. set address of clock table, baudclk, based on baud rate (0 thru 7)
; b. set table offset by baud rate;
; (these are standard CDQ2000 set by macro, BAUDCLK, in box_ct1.asm)
; 0 = 300 baud
; 1 = 1200 baud
; 2 = 2400 baud
; 3 = 3200 baud
; 4 = 4800 baud
; 5 = 38400 baud
; 6 = 9600 baud
; 7 = 19200 baud
; c. set transmit enable
; d. get and set the clock for baud rate from the table
; e. adjust to the sampling rate info
; f. get and set the max bytes for baud rate from the table

move    #baudclk,r0        ;get data baud rate table address
move    y:baudrte,b        ;set to access clock at baud rate
bset    #M_TE,x:<<M_SCR     ;set transmit enable
tst     b    #3,n0         ;test for rate of zero
                        ; & set register to advance thru table
jeq     <_baud_cds_        ;if code is zero, we're there

rep     b
move    (r0)+n0            ;position to selected baud rate code

_baud_cds_

move    y:(r0)+,r2         ;get clock value at baud rate
move    y:smplrte,n0       ;now get sampling rate offset
movep   r2,x:<<M_SCCR       ;set the clock for selected baud rate
move    y:(r0+n0),n1       ;get max byte count at sampling rate
move    n1,y:maxbytes      ;store maxbytes for scixmt to check

;set flags for sampling rate and type of data received

;!!!dbg
; move    y:frmrate,b
; tst     b

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-164-

```

;eq    <_bit_lo_          ;!!!dbg
SET_HI_BIT_RATE_DCD
jmp    <_smpl_

_bit_lo_
SET_LO_BIT_RATE_DCD

_smpl_
;!!!dbg
move   y:smplrte,b
tst    b          y:inputcde,b
;!!!dbg
jeq    <_type_
jeq    <_smpl_lo_        ;!!!dbg

bset   #SAMPLE_RATE_LOW_vs_HIGH,y:<ctrlflgs
SET_HI_SAMPLE_RATE_DCD
jmp    <_type_

;!!!dbg
_smpl_lo_
SET_LO_SAMPLE_RATE_DCD

_type_

;test for MUSICAM input data stream selected versus G722 data input stream

tst    b          ;0 = MUSICAM, else G722
jeq    <rate_      ;if 0, it's MUSICAM. test bit rate

g722_boot

;G722 input selected, signal the encoder XMICRMUS and
;boot up RMCRG722 from the low portion of chip

;!!!2/7/1994 SET_G722_DATA_DCD
bset   #MUSICAM_vs_G722,y:<ctrlflgs
OFF_FRAME_LED_DCD          ;douse the framing led alarm
OFF_CRC_ERROR_LED_DCD      ;douse the crc error led alarm
OFF_MONO_LED_DCD           ;douse the mono led indicator
OFF_JOINT_LED_DCD          ;douse the joint stereo led indicator
OFF_STEREO_LED_DCD         ;douse the stereo led indicator
OFF_LO_BIT_RATE_LED_DCD
OFF_HI_BIT_RATE_LED_DCD
ON_G722_LED_DCD            ;light the G722 front panel led
OFF_MUSICAM_LED_DCD
OFF_LO_SAMPLE_RATE_LED_DCD
OFF_HI_SAMPLE_RATE_LED_DCD
SET_LEDS_DCD               ;set the leds as needed
INTERRUPT_HOST_DCD
bclr   #11,x:<M_PBD        ;clr boot c000 for RMCRG722 boot c000
jmp    <bootup             ;boot in RMCRG722

rate_
;!!!dbg
SET_MUSICAM_DATA_DCD      ;!!!dbg

; setup synch variables

```

.165-

```

move    #outbuf,r7          ;setup synth variables
move    #2,n7               ;set to skip left and right
move    #OUTBUF-1,m7        ;set circular outbuf ctl
move    r7,r0               ;set up to set read and write ptrs
jsr     <alignptr           ;set ptrs

; Now set priorities of the IRQA and SSI peripherals
; IRQA priority = 0 turned off
; HOST set to IPL 2
; SSI priority = 2
; SCI priority = 2

movep   #>Sa000,x:<<M_IPR    ;set int priorities and edges
movep   #>Sa800,x:<<M_IPR    ;set int priorities and edges

;!!!debug tickle to see if chip booted
;_loop
;   bset    WATCH_DOG
;   bclr    WATCH_DOG
;   jmp     <_loop

;wait for the dust to settle before pushing onward

;KM      move    #>RDCDSYNT_STARTUP,a
;KM      jsr     <wait

        andi    #Sfc,mr      ;turn on the interrupt system

; NOW we are alive with interrupts on!

; Set the addresses of inbuf and nxbuf to receive the input data.

reframe

        bclr    #M_TIE,x:<<M_SCR    ;disable and data transmit interrupt
        CLR_DAC_RESET            ;clear the DAC reset line to mute output

;if G722 data input, go to the RMCRG722 boot-up routine
        jset     #MUSICAM_vs_G722,y:<<ctlflgs.g722_boot

;since it's musicam, keep in this routine and set indicators

        SET MUSICAM_DATA_DCD
        ON MUSICAM_LED_DCD
        OFF G722_LED_DCD
        ON_FRAME_LED_DCD          ;set the framing led alarm
        ON_CRC_ERROR_LED_DCD      ;set the crc error led alarm
        OFF_MONO_LED_DCD          ;set the mono led indicator
        OFF_JOINT_LED_DCD         ;set the joint stereo led indicator
        OFF_STEREO_LED_DCD        ;set the stereo led indicator

; set micro leds and indicators

        move     #framerate,r0
        nop
        jset     #0,y:(r0),do_hi_  ;test for frame higher Kbit rate
        SET_LO_BIT_RATE_DCD

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-166-

```

ON LO BIT RATE LED_DCD
OFF HI BIT RATE LED_DCD
jmp <_do_coding_

```

```

_do_hi_
SET HI BIT RATE DCD
ON HI BIT RATE LED_DCD
OFF LO BIT RATE LED_DCD

```

```

_do_coding_
jset #SAMPLE_RATE_LOW_vs_HIGH,y:<ctrlflgs,_hi_rte_ ;test hi sample
SET LO SAMPLE RATE DCD
ON LO SAMPLE RATE LED_DCD
OFF HI SAMPLE RATE LED_DCD
jmp <_do_pll_

```

```

_hi_rte_
SET HI SAMPLE RATE DCD
ON HI SAMPLE RATE LED_DCD
OFF LO SAMPLE RATE LED_DCD

```

```

_do_pll_
;check the phase lock loop signal:

```

```

TST SET PHASE LOCK_DCD,_set_PLL ;turn off phase lock led indicator
OFF PHASE LOCK_LED_DCD
jmp <_set_alm

```

```

_set_PLL
ON PHASE LOCK_LED_DCD ;turn on phase lock led indicator

```

```

_set_alm
ON ALARM_LED_DCD ;set alarm led indicator

```

```

TST SET ALARM_RELAY_DCD,_set_led_A ;unless already set,
SET ALARM_RELAY_DCD ;set the alarm relay line on

```

```

_set_led_A
SET LEDS_DCD ;set the leds as needed
INTERRUPT_HOST_DCD

```

```

;mute the audio output until we are framed

```

```

jsr <muteout ;mute the dac output buffer

```

```

;controls to force a reboot if an inordinate number of framing errors

```

```

move y:frtries,a ;get frame tries
move #>MAX_TRIES,x0 ;get number of tries tolerance
move #>3,x0 ;get number of tries tolerance
cmp x0,a #>1,y0 ;make test & set up to incr count
jge * ;kill watch dog, if reached tolerance
jlt <_dsb_dbg_

```

```

;if manual auto selection, do not force a reboot

```

```

move #autosel,r0
nop
jclr #0,x:(r0),_manual_restart ;manual select, do not reboot

```

-167-

```

        nop
        nop
        nop
        nop
        nop
        jmp      *           ;kill watch dog
        jmp      <restart    ;kill watch dog

_manual_restart
; if in manual mode, zero the failure counter

        clr      a
        move     a,y:frtries
        nop
        nop
        nop
        nop
        jmp      <restart    ;in manual mode start over

_dsb_dbg_
        add      y0,a      #syncbuf,r0    ;increment count of frames
                                           ; & get address of sync buffer
        move     a,y:frtries    ;update count of framing tries
        jsr      <framit       ;and frame the data

; test for successful framing, if not, restart

        tst      a          r3,y:IPbitoff  ;test if framed (a = 0 if framed)
                                           ; & save the bit offset
        jeq      <_ok_
        jne      <restart    ;NO, we must restart
        nop
        nop
        nop
        jmp      <restart

_ok_
; since we have MUSICAM frames, set the flag for auto select switches

        bset     #MUSICAM_INPUT_SET,y:<ctlflgs

; indicate to encoder that the decoder is framed and to use pins for:
; MUSICAM vs G722
; LOW vs HIGH sampling rate
; (otherwise, if auto selected and pin 14 is still low, encoder operates
; at MUSICAM at the LOW sampling rate)

        SET_DECODER_FRAMED_DCD

; initialize the polysynthesis arrays for the 1st frame

        jsr      <polysini

; the a reg is returned as 0 to go on
; clear the successive CRC-16 bit error sensed counter
; if exceeded according to the chkCRC routine, automatically reframe

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-168-

```

move    a,y:biterrs          ;zero the bit error counter
move    a,y:oof              ;zero out-of-frame faults counter
move    a,y:voof             ;zero sample rate code faults counter
move    a,y:pocf             ;zero CRC protection code faults counter
move    a,y:doof             ;0 ancil data errors/old CCS CRC-16 cnter
move    r5,y:IPwrdoff        ;save i/p buufer word offset
bclr    #FIRST_TIME,y:<ctlflgs ;clear the indicator
bclr    #FRAME_SAVED,y:<ctlflgs ;clear the indicator
bclr    #USE_SAVED,y:<ctlflgs  ;clear the indicator
bclr    #SAVE_FRAME,y:<ctlflgs ;clear the indicator
bclr    #USING_SAVED,y:<ctlflgs ;clear the indicator
bclr    #REFRAME,y:<ctlflgs    ;clear the indicator

OFF_FRAME_LED_DCD             ;douse decoder framed alarm led
SET_LEDS_DCD                 ;set the leds as needed
INTERRUPT_HOST_DCD

;for ancillary data decoding purposes, determine the end of the coded frame
jsr     <framend

;initialize the ancillary data controls for decoding and transmission
clr     a                    #databytes,r0 ;zero the decoded byte counter
; & get addr of the data byte buffer
move    a,y:bytecnt          ;bytes decoded counter set to zero
move    r0,y:dataiptr        ;address for next byte decoded
move    r0,y:dataoptr        ;addr for next byte to out RS232
dc      #DATABUFLEN,_clr_data
move    a,y:(r0)+            ;zero the ancillary data buffer

_clr_data
bset    #M_TIE,x:<<M_SCR      ;set the data transmit interrupt

; Let the show begin.
top
;get the external switches to determine if any changes that signal a restart
GET_SWITCHES_DCD gsws_20
jsr     <getsws
jset    #4,y:<not_appl, restart
jclr    #4,y:<not_appl,_ok_2_
nop
nop
nop
nop
nop
jmp     <restart
_ok_2_

;check the phase lock loop signal:
TST_SET_PHASE_LOCK_DCD,_set_ph
; if not set, clear the phase lock loop led and light the alarm led
CLR_DAC_RESET                ;clear the DAC reset line to mute output

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-169-

```

OFF_PHASE_LOCK_LED_DCD      ;turn off phase lock led indicator
ON_ALARM_LED_DCD            ;light alarm condition led indicator
TST_SET_ALARM_RELAY_DCD,_set_led_B
SET_ALARM_RELAY_DCD
jmp      <_set_led_B

```

_set_ph

```

; else, light the phase lock loop led
; and if there is no CRC bit error, clear the alarm led

```

```

ON_PHASE_LOCK_LED_DCD      ;light phase lock loop led indicator
TST_SET_CRC_ERROR_DCD,_set_alm_A ;if crc error set, turn alarm led on
OFF_ALARM_LED_DCD          ;turn off alarm led indicator

TST_CLR_ALARM_RELAY_DCD,_set_led_B
CLR_ALARM_RELAY_DCD
jmp      <_set_led_B

```

_set_alm_A

```

ON_ALARM_LED_DCD            ;light alarm condition led indicator

TST_SET_ALARM_RELAY_DCD,_set_led_B
SET_ALARM_RELAY_DCD

```

_set_led_B

```

OFF_OVERLOAD_LED_DCD      ;clear decoder overload alarm led
SET_LEDS_DCD              ;set the leds as needed
INTERRUPT_HOST_DCD

```

```

; bset WATCH_DOG      ;tickle the dog
; bclr WATCH_DOG

```

```

; Now wait until we have 1 word in the input buffer
; The variable waitform contains the address of one word after the sync word.
; This is the word to wait for in the interrupt routine to signal the
; start of a new frame.

```

```

move    y:frmmod,m0      ;set up m0 as a mod buffer of one frame
move    y:frmsize,n0     ;get buffer length
move    y:IPwrdoff,r0    ;word offset for frame start
move    y:frmsize,a      ;get 1/2 buffer length: frame length
lsl     a                ;times 2
move    a1,y0            ;set framing buf length for addr compare
move    (r0)+n0          ;increment to next input frame
move    r0,y:IPwrdoff    ;save new offset word to start of frame
move    (r0)+            ;increment 1 word
move    r0,x0            ;set as address to wait for
move    y:<linear,m0     ;restore r0 to linear addressing
move    y:frmsize,x1     ;get half the framing buffer size

```

```

; Here we check if we have received enough data to proceed
; This is done by checking by subtracting the

```

_rdec_15

```

; bset WATCH_DOG      ;tickle the dog
; bclr WATCH_DOG

move    y:<inpwptr,a     ;get curr read frames :/p ptr

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-170-

```

sub    x0,a           ;sub addr to wait for
jge    <_rdec_20      ;check for zero addr wrap around
add    y0,a           ;bump result by framing buffer length

_rdec_20:
cmp     x1,a           ;see if past a half a buffer
jlt     <_rdec_15      ;if not yet at the half-way, loop

;!!!DGCST
;!!!if required for even frame sizes when auto select sampling rate.
;!!!make sure no rate switch fooled the decoder
;
;    VERIFY_AUTO_SAMPLE    ;as needed by box_ctl.asm
;
;!!!DGCST

;take the next frame to decode and word align it for reed solomon decoding

move    y:IPwrdoff,r0    ;get the word offset for the next frame to decode
move    #synbuf,n0       ;base address of the i/p frame buffer
move    y:frmemod,m0     ;doubled buffer i/p
move    #reedsolbuf,r1   ;addr for Reed Solomon i/p buffer
move    #framebuf,r2     ;addr for MUSICAM decode frame i/p buffer
move    (r0)+n0          ;get to start addr of current i/p frame
move    y:frmsize,n0     ;number of words in a frame
move    y:IPbitoff,b     ;bit offset to sync pattern in 1st word

;for the length of a full frame.
;    get the words in pairs and shift to word boundary

do      n0,_reed_shift
move    x:(r0)+,a1       ;1st word of the curr pair to shift

;if words already are aligned, simply copy the word to the Reed Solomon buffer

test    b    x:(r0),a0    ;see if a shift is needed.
;    & get 2nd word of curr pair to shift
jeq     <_no_shift       ;if no offset, no shift needed

;for the number of offset bits, shift the pair of words to abut properly aligned

rep     b
asl     a

_no_shift

;copy aligned word in Reed Solomon buffer for decoding

;!!!dbg
move    a1,x:(r1)+
move    a1,x:(r2)+       ;also copy to MUSICAM frame buffer

_reed_shift:
;decode the Reed Solomon frame back to a MUSICAM frame

move    y:<linear,m0     ;restore r0 to linear addressing
move    #reedsolbuf,r6   ;Reed Solomon frame buffer: i/p
move    #framebuf,r1     ;frame buffer decoded: o/p
move    #PRCF1,r3        ;Reed Solomon profile: control decode

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL 

-171-

```

        jsr      <rsdec16          ;do Reed Solomon decode

; Now setup the buffer reading routines

        move     y:dcdfmod,m6      ;decoded Reed Sol frame bufmod ctrl
        move     #framebuf,n6     ;decoded Reed Solomon frame buffer addr
        move     y:wrdoff,r6      ;bit offset from msb
        move     y:bitoff,a       ;bit offset from msb

        bclr     #USE_SAVED,y:<ctlflgs ;clear used saved frame flag
        bclr     #USING_SAVED,y:<ctlflgs ;clear using saved frame flag

        OFF_CRC_ERROR_LED_DCD      ;turn off the crc error led indicator
        TST_SET_PHASE_LOCK_DCD,_clr_alm_A ;if not phase loop locked, then

        CLR_DAC_RESET              ;clear the DAC reset line to mute output

        ON_ALARM_LED_DCD           ;light alarm led indicator
        TST_SET_ALARM_RELAY_DCD,_set_led_C
        SET_ALARM_RELAY_DCD        ;turn the alarm relay on
        jmp      <_set_led_C

_clr_alm_A
;release the digital to analog converter for output

        SET_DAC_RESET              ;set the DAC reset line high now

        OFF_ALARM_LED_DCD          ;turn off alarm led indicator
        TST_CLR_ALARM_RELAY_DCD,_set_led_C
        CLR_ALARM_RELAY_DCD        ;turn the alarm relay off

_set_led_C
        SET_LEDS_DCD               ;set the leds as needed
        INTERRUPT_HOST_DCD

        bclr     #SAVE_FRAME,y:<ctlflgs ;clr ind for getvalue to save frame wds

;Now we are ready to decode the current frame using:
; n6 = buffer address
; r6 = word offset into the buffer for start of the frame
; a = bit offset into the word offset into the buffer for start of the frame
; m6 = mod buffer control through the buffer this will be either
;      normal input for 3 * frame size -1 (leaves space for saved buffer)
;      single frame size -1 for using the saved frame if a checksum error)

_rdec_30

        bset     WATCH_DOG        ;tickle the dog
        bclr     WATCH_DOG        ;tickle the dog

        TOGGLE_WATCH_DOG_DCD

        jsr      <bitsall0

;prepare to suppress ancillary data if any out of frame condition

        bclr     #NO_SYNC,y:<ctlflgs ;clear the indicator

; Now get the sync pattern. If the pattern matches a good sync, then

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-172-

; the oof counter is decremented. If it doesn't match, the oof pattern
; is incremented. If it is incremented past an upper limit, an out of
; frame condition is declared and the system goes into framing.
; On the other hand, the oof counter is never allowed to go negative.

```

jsr    <getsync          ;get the sync bits
move   a1,y0             ;move right justified value
move   y:cof,b           ;get current # of oof's

; if using the saved frame, do not recount sync problems

jset   #USE_SAVED,y:<ctlflgs,_rdec_50
move   #>SYNC,a          ;get sync pattern for test
cmp    y0,a    #>GOOD_DECREMENT,x1 ;do we have a valid sync
                                ; & set good sync decrement value
jeq    <_rdec_40

```

; We are here because the sync did not match.
; Increment the number of bad syncs found.

```

bset   #NO_SYNC,y:<ctlflgs ;set indicator to skip ancillary data

move   #>BAD_INCREMENT,x1 ;set the bad match increment value
add    x1,b    #>BAD_LIMIT,x0 ;increment the number of oof's
                                ; & set limit value to restart
cmp    x0,b
jlt    <_rdec_50           ;we are not, so keep going

nop
nop
nop
nop
nop

```

; we've sensed too many sync pattern failures in succession

TOO_MANY_SYNC_ERRORS_DCD

```

!!!micromus    jmp    <restart          ;at error limit so reframe

```

; We are here because a valid sync was found.
; Decrement the number of bad syncs found.

```

_rdec_40
sub     x1,b             ;decrement the number of oof's
tst     b    #0,x1       ;see if at the limit
tlt     x1,b

```

```

_rdec_50
move    b,y:oof         ;save the current oof counter

```

; get the system header info

```

jsr     <getsyst        ;get system header info

```

; see if the frame header sample rate code matches determined sampling rate
; If the sample rate codes match a good sync, then the vooof counter is
; decremented.
; If the codes don't match, the vooof counter is incremented.

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-173-

If the voof counter is incremented past an upper limit, we have to do the auto selection again since perhaps the sampling rate has changed.

```

move    y:svesmpl,a          ;get code from frame header
move    y:smpicde,x0         ;get code determined by framing
move    y:voof,b             ;get current # of voof's
cmp      x0,a    #>GOOD_DECREMENT,x1    ;is a valid sample rate code
; & set good code decrement value
jne      <_ck_smp1_05        ;if we don't that's bad

```

now check the frame header ID that matches the sample rate

```

move    y:sveidbit,a         ;get ID from frame header
move    y:smpidbit,x0        ;get ID determined by framing
cmp      x0,a                ;see if a match
jeq      <_ck_smp1_10        ;if we do that's good

```

_ck_smp1_05

; We are here because there was no match of the sample rate codes.
; Increment the number of unmatched sample rate codes.

```

move      #>BAD_INCREMENT,x1    ;set the bad match increment value
add       x1,b    #>BAD_LIMIT,x0 ;increment the number of voof's
; & set limit value to restart
cmp       x0,b
jlt       <_ck_smp1_20          ;see if at the limit
; we are not, so keep going
;!!!dbg
nop
nop
nop
nop
nop
;!!!dbg
jmp       <restart              ;at error limit so restart

```

; We are here because a valid sample rate was found in the frame header.
; Decrement the number of unmatched sample rate codes.

```

_ck_smp1_10
sub       x1,b
tst       b    #0,x1           ;decrement the number of voof's
; see if at the limit
tlt       x1,b                ;if less than zero, set to zero

```

```

_ck_smp1_20
move      b,y:voof             ;save the current voof counter

```

; see if the frame header CRC protection code matches determined protection code
; If the codes match, then the poof counter is decremented.
; If the codes don't match, the poof counter is incremented.
; If the poof counter is incremented past an upper limit, we have to
; do the auto selection again since perhaps the CRC protection has changed.

```

move      y:poof,b             ;get current # of poof's
move      #>GOOD_DECREMENT,x1  ;set good match decrement value

```

verify the CRC PROTECT setting versus auto sampling:
if the frame header shows CRC protection,
verify auto sample also indicates protection

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-174-

```

;set    #PROTECT,y:<ctiflgs,_ck_prot_00 ;if protect, check auto
;frame shows no protection.
; if auto sampling also found no protection.
;   go to decrement the poof counter
; otherwise, force protection and assume a bit error
;   and increment the poof counter

;set    #0,y:<protect,_ck_prot_10      ;if match, decrement poof
;set    #PROTECT,y:<ctiflgs           ;set the CRC applies bit
;mp     <_ck_prot_05                  ;go to increment poof for the bad match

_ck_prot_00
;frame shows protection.
; if auto sampling also found protection, continue
; otherwise, force no protection and assume a bit error
;   and increment the poof counter

;clr    #0,y:<protect,_ck_prot_10      ;if match, decrement poof
;clr    #PROTECT,y:<ctiflgs           ;clear the CRC applies bit

_ck_prot_05
; We are here because there was no match of the CRC protection codes.
; Increment the number of unmatcheds found.

move    #>BAD_INCREMENT,x1           ;set the bad match increment value
add     x1,b    #>BAD_LIMIT,x0       ;increment the number of poof's
; & set limit value to restart
cmp     x0,b
jlt     <_ck_prot_20                 ;see if at the limit
;we are not, so keep going
;!!!dbg
nop
nop
nop
nop
nop
;!!!dbg
jmp     <restart                      ;at error limit so restart

; We are here because a valid CRC protection code was found in the frame header.
; Decrement the number of unmatched CRC protection codes.

_ck_prot_10
sub     x1,b                          ;decrement the number of poof's
tst     b    #0,x1                   ;see if at the limit
tlt     x1,b                          ;if less than zero, set to zero

_ck_prot_20
move    b,y:poof                     ;save the current poof counter

;if there is CRC-16 protection on the frame:
; set the CRC-16 checksum bit count for the old ISO method:
; a. header bits covered by any type of frame
;   plus bits for the left channel also apply to any type of frame
; b. set bits for possible right channel based on frame type
; c. if not MONG, add bits for right channel
; d. save old ISO bit count for this frame

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL

-175-

```

jclr    #PROTECT,y:<ctlflgs,_rdec_60    ;if no checksum, get allocations
move    #>CRC_BITS_A-CRC_BITS_B,a
move    #>CRC_BITS_B,x0                ;bit count for right channels
jset    #STEREO_vs_MONO,y:<ctlflgs,_rdec_52
add     x0,a                          ;since its stereo, add for right channel
_rdec_52
move    a,x:crcold                    ;set the old ISO CRC-16 bit count
bset    WATCH_DOG                     ;tickle the dog
bclr    WATCH_DOG                     ;tickle the dog
jsr     <getcrc                        ;get checksum from frame
_rdec_60
move    #SBIndx,r0                    ;address of sub-band indicies
jsr     <getbal                        ;get bit allocations
move    #SBits,r0                     ;address of SB bits array
move    #SBIndx,r1                    ;address of sub-band indicies
jsr     <getsbits                      ;get the sb bits
move    #SBndsSKF,r0                  ;address of the SB scale factors
move    #SBits,r1                     ;address of SB bits array
move    #SBIndx,r2                    ;address of sub-band indicies
jsr     <getskf                       ;get scale factors
jclr    #PROTECT,y:<ctlflgs,_rdec_70    ;if no checksum, get data pts
;!!!dbg
; jmp    <_rdec_70
;!!!dbg
; bset    WATCH_DOG                  ;tickle the dog
; jset    #USE_SAVED,y:<ctlflgs,_rdec_70 ;do not recheck saved frame
; jsr     <chkcrc                     ;check the validity of frame
; jset    #REFRAME,y:<ctlflgs,reframe   ;if too many bit errors, reframe
; jclr    #REFRAME,y:<ctlflgs,_dbg_dsb_ ;if too many bit errors, reframe
; nop
; nop
; nop
; nop
; TOO_MANY_BIT_ERRORS_DCD
_dbg_dsb
jclr    #USE_SAVED,y:<ctlflgs,_rdec_65 ;if valid, continue with frame
jclr    #USING_SAVED,y:<ctlflgs,_rdec_65 ;if saved valid, continue
ON_CRC_ERROR_LED_DCD                ;light crc error alarm led
ON_ALARM_LED_DCD                    ;light alarm led indicator
; TST SET_ALARM_RELAY_DCD,_set_led_D
; SET_ALARM_RELAY_DCD                ;turn the alarm relay on
_set_led_D
SET_LEDS_DCD                        ;set the leds as needed
INTERRUPT_HOST_DCD
jclr    #FRAME_SAVED,y:<ctlflgs,_rdec_80 ;else failed, if no saved frame

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-176-

```

                                : output zeroes and try again
                                : clear since we used the saved frame
                                : else, set up last saved frame
                                : word offset was saved
                                : bit offset was saved
bclr    #FRAME_SAVED,y:<ctrlflgs
move    #savebuf,n6
move    y:wrdooff,r6
move    y:bitoff,a
                                : go back and do last frame again
jmp     <_rdec_30

_rdec_65
OFF_CRC_ERROR_LED_DCD          : turn off the crc error alarm led
                                : tickle the dog
bclr    WATCH_DOG

_rdec_70
;now, light the proper led for the type of framing:
;full stereo, joint stereo, dual channel or mono

jset    #STEREO vs MONO,y:<ctrlflgs,_rdec_53    ;if mono
jset    #JOINT_FRAMING,y:<ctrlflgs,_rdec_51     ;if joint stereo

OFF_MONO_LED_DCD              : turn off the mono led indicator
OFF_JOINT_LED_DCD            : turn off the joint stereo led indicator
ON_STEREO_LED_DCD            : light the stereo led indicator
jmp     <_rdec_55

_rdec_51
OFF_MONO_LED_DCD              : turn off the mono led indicator
OFF_STEREO_LED_DCD           : turn off the stereo led indicator
ON_JOINT_LED_DCD             : light the joint stereo led indicator
jmp     <_rdec_55

_rdec_53
OFF_STEREO_LED_DCD           : turn off the stereo led indicator
OFF_JOINT_LED_DCD           : turn off the joint stereo led indicator
ON_MONO_LED_DCD             : light the mono led indicator

_rdec_55
SET_LEDS_DCD                  : set the leds as needed
INTERRUPT_HOST_DCD

;test if the fade controls are applicable

TST_CLR_FADE_OUTPUT_DCD,_fade_5 ;if fade not requested, continue
move    y:fadecnt,b           ;get fade frame counter
tst     b                      ;test if ready to fade (fadecnt=0)
                                ; & set to decrement frame count
                                ;not ready yet, go decrement
jne     <_fade_3               ;get current fade value
move    y:fade,a              ;get maximum fade down range
move    #>FADE_SOFTTEST,y0    ;increment to soften output
TST_SET_FADE_DOWN_DCD,_fade_1 ;increment to soften output
tst     a                      ;test if at loudest fade up
                                ; & get test for max start fade value
                                ;if at loudest, continue
jeq     <_fade_5               ;test if above max start fade
cmp     x1,a                   ; & get scale factor increment
                                ;if needed, set start fade up
tgt     x1,a                   ;adjust louder for this frame
sub     x0,a                   ; & set frame count to next decrement
                                ; & set frame count to next decrement

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-177-

```

        jmp      <_fade_2          ;store new fade SKF adjust value

_fade_1  cmp      y0,a      #>FADE_INCREMENT,x0
        jeq      <_fade_5          ;if at softest, continue
        add      x0,a      #>FADE_FRAMES,b ;adjust softer for this frame
                                         ; & set frame count to next decrement

_fade_2  move     a,y:fade          ;save the new fade SKF adjust value
        jmp      <_fade_4

_fade_3  sub      x0,b              ;decrement frame counter

_fade_4  move     b,y:fadecnt       ;save the new fade frame counter

_fade_5

; if 1st frame align the ptrs for the polysynthes
        jset     #FIRST_TIME,y:<ctlflgs,_rdec_57
        move     r7,r0              ;align the read & write ptrs
        jsr      <alignptr          ;set ptrs
        bset     #FIRST_TIME,y:<ctlflgs ;indicate ptrs have been aligned

_rdec_57
        move     #SBIdx,r3          ;sb indicies
        move     #SBndSKF,r2        ;get the scale factors
        move     #ASMDData,r1       ;set A share mem of rec samples
        jsr      <getdata           ;get the sub-band data

        jsr      <getancdata        ;process ancillary data

; maintain the frame counter of successive frames with the old CCS CRC-16
; checksum coupled with ancillary data decoding problems.
; If the no error was detected, then the doof counter is decremented.
; If there was an error, the doof pattern is incremented. If it is
; incremented past an upper limit, an out of frame condition is declared
; and the system may go into reframing swapping the old CCS decoding for
; MPEG-ISO decoding or vice versa.
; The doof counter is never allowed to go negative.

        move     y:doof,b           ;get current # of doof's

; A saved frame is not included in maintaining the doof's counter.

        jset     #USE_SAVED,y:<ctlflgs,_rdec_150

; check if a problem with old CCS CRC-16 algorithm coupled with
; a problem with ancillary data.

        move     #oldccs,r1          ;addr to test ancillary data problem
        move     #>GOOD_DECREMENT,x1 ;to decrement error frame counter
        jclr     #2,y:(&1),_rdec_140 ;if no ancillary data error, decrement

; We are here because there was an ancillary data problem/old CCS CRC-16
; increment the number of bad frames found.

```



-178-

```

move    #>BAD_INCREMENT,x1    ;to increment the number of doof's
add     x1,b    #>BAD_LIMIT,x0 ;increment the number of doof's
                                ; & set limit value to restart
cmp     x0,b     ;see if at the limit
jlt     <_rdec_150 ;we are not, so keep going
;!!!dbg
nop
nop
nop
nop
nop
;!!!dbg
;reframe if too many ancillary data problems in succession
TOO_MANY_DATA_ERRORS_DCD
jmp     <_rdec_150

; We are here because the ancillary data decoded ok
; Decrement the number of ancillary data problem frames found
_rdec_140
sub     x1,b     ;decrement the number of doof's
tst     b        #0,x1 ;see if at the limit
slt     x1,b     ;if less than zero, set to zero
_rdec_150
move    b,y:doof ;save the current doof counter
;clr    #PROTECT,y:<ctrlflgs,_rdec_72 ;if no checksum, no reason to save
;clr    #USE_SAVED,y:<ctrlflgs,_rdec_72 ;did not use a saved frame
;do not reuse a saved frame
bclr    #FRAME_SAVED,y:<ctrlflgs ;clear we have a saved frame flag
jmp     <top
_rdec_72
;since we had a good new frame, check controls for long solid operation
; restart the counter of frames with bit error
; and adjust count of framing retries, that control reset needed
clr     b        #>1,y0 ;zero bit successive bit error counter
                                ; & to decrement counter every frame
move    y:frtries,a ;get framing try counter
sub     y0,a    b,y:biterrs ;decrement counter every frame
                                ; & zero bit error counter
tst     a     ;see if counter reached zero
jge     <_rdec_75 ;if not, continue
clr     a     ;zero framing tries
_rdec_75
move    a,y:frtries ;save the reduced framing tries ctr
jmp     <top ;do next frame
_rdec_80
OFF_MONO_LED_DCD ;turn off the mono led indicator
OFF_JOINT_LED_DCD ;turn off the joint stereo led indicator

```

-179-

OFF_STEREO_LED_DCD
SET_LEDS_DCD
INTERRUPT_HOST_DCD

;turn off the stereo led indicator
;set the leds as needed

;mute the current frame

jsr <muteout
jmp <top
end start

;mute the output buffer

BAD ORIGINAL 

SUBSTITUTE SHEET (RULE 26)

-180-

```

opt      fc.

; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \URDCDSYN\getsbits.asm: Ben's mux
;
; title   'Get SB bits'
;
; This routine is used to get the SB bits of each of the sub-bands.
;
; on entry
;   r0 = address of the bit SB array
;   r1 = address of the SubBandIndex array
;
;   r6 = current offset in the input array
;   n6 = base address of the input array
;   y:<maxsubs = MAXSUBBANDS at sampling rate and bit rate
;   y:sc = shift count of current input word
;   x:crcbits = accumulator of bits covered by CRC-16 routine
;               (bit coded for SBits are accumulated)
;
; on exit
;   r6 = updated
;   y:sc = updated
;
;   a = destroyed
;   b = destroyed
;   x0 = destroyed
;   x1 = destroyed
;   y0 = destroyed
;   y1 = destroyed
;   r0 = destroyed
;   r1 = destroyed
;   r4 = destroyed
;   n4 = destroyed
;
; include 'def.asm'
;
; org      phe:
;
; initialize:
;   a. number of frame bits for a sub-band SBits index value
;   b. n0 offset for right channel sub-band SBits values:
;       left channel from 0 to (NUMSUBBANDS - 1)
;       right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)
;   c. n1 offset for right channel sub-band bit allocation values:
;       left channel from 0 to (NUMSUBBANDS - 1)
;       right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)
;
; getsbits
;   move     #NSBITS,n4           ;set number of bits to get
;   move     #NUMSUBBANDS,n0      ;SBits offset-right channel
;   move     #NUMSUBBANDS,n1      ;bit alloc offset-right channel
;   move     x:crcbits,r5         ;get CRC-16 bit counter
;   move     n4,n5               ;to accumulate CRC-16 bits
;
; loop through the sub-bands extracting the left and right (if applicable)
; SBit values values (y:<maxsubs = fixed count of sub-bands framed):
; process the right channel:
;   a. fcr current sub-band get the left channel allocation index value

```

SUBSTITUTE SHEET (RULE 26)

-181-

```

; b. if the left channel index is zero, go to insert a zero SBits value
; c. otherwise, extract the SBits value for left channel of current sub-band
; and go to insert value into the SBits array

do      y:<maxsubs,_gets_90
move    x:(r1),b
tst     b
jeq     _gets_10
jsr     getvalue
move    #>MASKNSBITS,x1
and     x1,a      (r5)+n5
jmp     _gets_20
;get left index for subband
;test index for not coded (0)
;use value of zero if not
;get a sb value
;mask for sbits from getvalue
;mask off hi order one's
; & accum bits for CRC-16 rtn
;go to store SBits value

;insert 0 for the left channel SBits value for this sub-band
_gets_10
clr     a
;no index use zero
;move the left channel SBits value to the SBits array
_gets_20
move    a1,x:(r0)
;process the right channel:
; a. for current sub-band get the right channel allocation index value
; b. if the right channel index is zero, go to insert a zero SBits value
; c. otherwise, extract the SBits value for right channel of current sub-band
; and go to insert value into the SBits array

move    x:(r1+n1),b
tst     b
jeq     _gets_30
jsr     getvalue
move    #>MASKNSBITS,x1
and     x1,a      (r5)+n5
jmp     _gets_40
;get right index for subband
;test index for not coded (0)
;use value of zero if not
;get a sb value
;mask for sbits from getvalue
;mask off hi order one's
; & accum bits for CRC-16 rtn
;go to store SBits value

;insert 0 for the right channel SBits value for this sub-band
_gets_30
clr     a
;no index use zero
;move the right channel SBits value to the SBits array
;increment SBits array and bit allocation index arrays for next sub-band
_gets_40
move    a1,x:(r0+n0)
move    (r0)+
move    (r1)+

_gets_90
move    r5,x:crcbits
;store updated CRC-16 bit counter

rts

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-182-

```

opt      fc,mex

; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \URDCDSYN\getskf.asm: Ben's mux

        title   'Get Scale Factors'

; This routine is used to get the scale factors of each of the sub-bands.

; on entry
;   r0 = address of the bit scale factor array (x memory)
;   r1 = address of the bit SB array (x memory)
;   r2 = address of the bit SubBandIndex array (x memory)
;
;   r6 = current offset in the input array
;   n6 = base address of the input array
;   y:<maxsubs = MAXSUBBANDS at sampling rate and bit rate
;   y:sc = shift count of current input word

; on exit
;   r6 = updated
;   y:sc = updated
;
;   a = destroyed
;   b = destroyed
;   x0 = destroyed
;   x1 = destroyed
;   y0 = destroyed
;   y1 = destroyed
;   r0 = destroyed
;   r4 = destroyed
;   n4 = destroyed

        include 'def.asm'
        include 'box_ctl.asm'

org      phe:

getskf

; initialize:
;   number of frame bits for a sub-band scale factor index value

        move     #SKF,n4                ;set number of bits to get
        move     #0,n0                  ;scale facts offset-left chan

; test the scale factors for certain tolerances:
;   a. zero scale factor is equivalent to a bit error,
;       indicate NO zero scale factor
;   b. clear the channel overload led indicators

        bclr     #SKF_ZERO,y:<ctlflgs
        OFF_LEFT_OVER_LED_DCD
        OFF_RIGHT_OVER_LED_DCD

; loop through the sub-bands extracting the left and right (if applicable)
; scale factor index values (y:<maxsubs = fixed count of sub-bands framed);
; within the sub-band loop is a loop for both channels: left then right

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-183-

```

; process the left channel:
; a. n0 offset for left channel sub-band scale factor index values:
;   left channel from 0 to (NUMSUBBANDS*NPARGROUP - 1)
;   right channel from NUMSUBBANDS*NPARGROUP
;       to ((2 * NUMSUBBANDS*NPARGROUP) - 1)
; b. n1 offset for left channel sub-band SBits values:
;   left channel from 0 to (NUMSUBBANDS - 1)
;   right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)
; c. n2 offset for left channel sub-band bit allocation values:
;   left channel from 0 to (NUMSUBBANDS - 1)
;   right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)

do      y:<maxsubs,_gets_90
move    #0,n1                                ;SBits offset-left channel
move    #0,n2                                ;bit alloc offset-left channel
bclr    #LEFT_vs_RIGHT,y:<ctiflgs           ;left is current channel

; process a channel for the current sub-band: 1st left then right
; a. update the register pointer with the offset into the scale factor
;   index array for the left or right channel
; b. get the bit allocation for the proper channel to see if any factors at all

do      #NUMCHANNELS,_gets_80
move    (r0)+n0                                ;offset for proper channel
move    x:(r2+n2),a                            ;get the SubBandindex[SubBand]

; first check if sub-band contains anything to work on. This value could
; be zero if there is no energy in the sub-band.

tst     a      x:(r1+n1),a                    ;see if any allotted bits
jne     _gets_05                             ;there were

; no bits were allocated, so set the scale factors to 63. I could just
; set the scale factors to anything for this case, but I set them to the
; lowest (actually, 63 is one lower than the lowest) scale factor.

move    #>63,a1                                ;get lowest scale factor value
move    a1,x:(r0)+
move    a1,x:(r0)+
move    a1,x:(r0)+
jmp     _gets_40

_gets_05
tst     a      #>1,x0                          ;SB == 0 for this sub-band
jne     _gets_10                             ; set x0 to sbit code '01'

; sbit code '00' case where must get all 3 scale factors

do      #3,_gets_a
jsr     getvalue
move    #>MASKSKF,x1                          ;mask for scale factor hi ord
and     x1,a                                    ;mask off high order one's
move    a1,x:(r0)+                            ;save in SubBandSKFs[SubBand][2]

_gets_a
jmp     _gets_40

_gets_10

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-184-

```

cmp      x0,a      #>3,x0      ;SB == 1 for this sub-band
jne      _gets_20      ; set x0 to sbit code '11'

; sbit code '01' case where must get the second two scale factors

jsr      getvalue
move     #>MASKSKF,x1      ;get SubBandSKFs[SubBand][1]
and      x1,a      ;mask for scale factor hi ord
move     a1,x:(r0)+      ;mask off high order one's
move     a1,x:(r0)-      ;save in SubBandSKFs[SubBand][0]
jsr      getvalue
move     #>MASKSKF,x1      ;get SubBandSKFs[SubBand][2]
and      x1,a      ;mask for scale factor hi ord
move     a1,x:(r0)-      ;mask off high order one's
                        ;save in SubBandSKFs[SubBand][2]

jmp      _gets_40

_gets_20
cmp      x0,a      #>2,x0      ;SB == 3 for this sub-band
jne      _gets_30      ; set x0 to sbit code '10'

; sbit code '11' case where must get the first two scale factors

jsr      getvalue
move     #>MASKSKF,x1      ;get SubBandSKFs[SubBand][0]
and      x1,a      ;mask for scale factor hi ord
move     a1,x:(r0)+      ;mask off high order one's
jsr      getvalue
move     #>MASKSKF,x1      ;get SubBandSKFs[SubBand][1]
and      x1,a      ;mask for scale factor hi ord
move     a1,x:(r0)+      ;mask off high order one's
move     a1,x:(r0)-      ;save in SubBandSKFs[SubBand][1]
                        ;save in SubBandSKFs[SubBand][2]

jmp      _gets_40

_gets_30
cmp      x0,a      ;SB == 2 for this sub-band
jne      _gets_40

; sbit code '10' case where must get the first factor

jsr      getvalue
move     #>MASKSKF,x1      ;get SubBandSKFs[SubBand][0]
and      x1,a      ;mask for scale factor hi ord
move     a1,x:(r0)+      ;mask off high order one's
move     a1,x:(r0)-      ;save in SubBandSKFs[SubBand][0]
move     a1,x:(r0)-      ;save in SubBandSKFs[SubBand][1]
                        ;save in SubBandSKFs[SubBand][2]

; set up for the right channel:
; a. backup the SKFs array for the left channel 3 scale factors indices
; b. n0 offset for right channel sub-band scale factor index values:
;    left channel from 0 to (NUMSUBBANDS*NPARGROUP - 1)
;    right channel from NUMSUBBANDS*NPARGROUP
;    to ((2 * NUMSUBBANDS*NPARGROUP) - 1)
; c. n1 offset for right channel sub-band SBits values:
;    left channel from 0 to (NUMSUBBANDS - 1)
;    right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-185-

```

; d. n2 offset for right channel sub-band bit allocation values:
;   left channel from 0 to (NUMSUBBANDS - 1)
;   right channel from NUMSUBBANDS to ((2 * NUMSUBBANDS) - 1)
;
_gets_40
;back up for the 3 scale factors and while doing it test for:
;  a. zero scale factor
;  b. overload scale factor

      move    y:fade,y1                      ;get current fade value
      do      #NPERGROUP,_gets_40_e
      move    x:-(r0),a
      add     y1,a    #>63,y0                ;apply scale factor fade
                                          ; & set maximum scale factor
      tst     a    #>OVERLOAD_SKF,x0
      jne     _gets_40_a
      bset    #SKF_ZERO,y:<ctrlflgs         ;1/4/94 do not set bit error
      move    y0,a                          ;1/4/94 set scale factor to 63
      jmp     _gets_40_d

; test for an overload, and if so, set channel led
_gets_40_a
      cmp     x0,a
      jge     _gets_40_c                    ;NO overload, test for max

;overload sensed, set which channel led
      jset    #LEFT_vs_RIGHT,y:<ctrlflgs,_gets_40_b
      ON_LEFT_OVER_LED_DCD
;!!!dbg
      nop
      nop
      nop
      nop
      nop
;!!!dbg
      jmp     _gets_40_c                    ;test for max SKF

_gets_40_b
      ON_RIGHT_OVER_LED_DCD
;!!!dbg
      nop
      nop
      nop
      nop
      nop
;!!!dbg

_gets_40_c
      cmp     y0,a                          ;test if greater 63
      jle     _gets_40_d                    ;if less or eq, use current
      move    y0,a                          ;if so, set to 63

_gets_40_d
      move    a,x:(r0)                      ;restore scale factor

```

SUBSTITUTE SHEET (RULE 26)

-186-

```

_gets_40_e
    bset    #LEFT vs RIGHT,y:<ctrlflgs    ;indicate current channel
    move    #NUMSUBBANDS*NPARGROUP,n0      ;scale facts offset-right chan
    move    #NUMSUBBANDS,n1                ;SBits offset-right channel
    move    #NUMSUBBANDS,n2                ;bit alloc offset-right channel
;after processing the right channel, set up for the left channel of the
; next sub-band:
; a. reincrement r0 for scale factor array by 3 for the inserted 3 factors
; b. to reposition the scale factor index array from right back to left channel
;    we put the negative offset in n0
; c. increment the SBits value array for the next sub-band
; d. increment the bit allocation index array for the next sub-band
_gets_80
    move    #3,n0
    move    (r1)+
    move    (r2)+
    move    (r0)+n0
    move    #-NUMSUBBANDS*NPARGROUP,n0      ;scale facts offset-right chan
_gets_90
    SET_LEDS_DCD                            ;show overload conditions
    rts

```

-187-

opt fc.mex

```

; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \URDCDSYN\getsws.asm

```

```

title 'Get decoder external switch settings'

```

```

; This routine is used to interpret the external switches on the box

```

```

; on exit

```

```

; x:tstrate = raw bit rate input from the switches
; x:tstsel1 = raw application of line 1 select switch
; x:tstsel2 = raw application of line 2 select switch
; x:tstfrmt = frame communication formatting
; x:tstreed = Reed/Solomon encoding switch
; x:tstbaud = raw ancillary data baud rate input from the switches

```

```

; y:<not_appl = bit 4 set if any switches changed

```

```

destroyed:

```

```

register a

```

```

include 'def.asm'

```

```

include 'box_ctl.asm'

```

```

section highmisc

```

```

xdef select1 ;current setting of line 1 select switch
xdef select2 ;current setting of line 2 select switch
xdef tstrate,tstsel1,tstsel2,tstfrmt,tstreed,tstbaud,tstmeth

```

```

org xhe:

```

```

stgetsws_xhe

```

```

select1 ds 1 ;current setting of line 1 select switch
select2 ds 1 ;current setting of line 2 select switch
tstrate ds 1 ;raw bit rate input from the switches
tstsel1 ds 1 ;raw application of line 1 select switch
tstsel2 ds 1 ;raw application of line 1 select switch
tstfrmt ds 1 ;raw frame communication formatting
tstreed ds 1 ;Reed/Solomon encoding switch
tstbaud ds 1 ;raw ancil data baud rate input from switches
tstmeth ds 1 ;raw code for diagnostic vs normal operation

```

```

endgetsws_xhe

```

```

endsec

```

```

org phe:

```

```

getsws

```

```

bclr #4,y:<not_appl ;indicate no changes initially

```

```

clr a

```

```

move a,x:tstrate

```

```

move a,x:tstsel1

```

```

move a,x:tstsel2

```

```

move a,x:tstfrmt

```

```

move a,x:tstreed

```

```

move a,x:tstbaud

```



-188-

```

move    a,x:tstmeth

;check the dip switches to determine frame bit rate
; and ancillary data application and data baud rate

;switches for framing bit rate

GET_BIT_RATE_DCD

;switches for framing type code and mono output

GET_FRAME_TYPE_DCD

;switches to set if selecting line 1 and/or line 2

GET_SELECTED_LINES_DCD

;switches for ancillary data baud rate

GET_BAUD_RATE_DCD

;switches for method of operation, normal audio or diagnostics

GET_DIAGNOSTICS_DCD

move    x:tstrate,y1          ;look for a change in framing rate
move    y:rawrate,a
cmp     y1,a    x:tstsel1,y1   ;set up to test line 1 selection
jne     _gsws_80
move    x:select1,a
cmp     y1,a    x:tstsel2,y1   ;set up to test line 2 selection
jne     _gsws_80
move    x:select2,a
cmp     y1,a    x:tstfrmt,y1   ;set up to test framing format
jne     _gsws_80
move    y:frmformat,a
cmp     y1,a    x:tstreed,y1   ;set up to test Reed/Solomon switch
jne     _gsws_80
move    y:reedsolomon,a
cmp     y1,a    x:tstbaud,y1   ;set up to test ancillary data baud
jne     _gsws_80
move    y:baudrate,a
cmp     y1,a
jne     _gsws_80

;see if we have to switch from normal to the diagnostic method of operation

move    x:tstmeth,a          ;get the diag nostic code
tst     a                   ;see if other than normal operation
jeq     _gsws_90            ;normal operation, continue

_gsws_80
bset    #4,y:<not_appl       ;indicate changes in external switches

_gsws_90
rts

```

BAD ORIGINAL

-189-

```
opt    fc,mex
;
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
;
; \URDCDSYN\getsync.asm: Ben's mux
;
; title    'Get Sync'
; This routine gets the sync word.
; on exit
;   a1 = right justified sync value padded on right with zeros
;   r6 = updated
;   y:sc = updated
;
;   a2 = destroyed
;   a1 = destroyed
;   b = destroyed
;   x0 = destroyed
;   x1 = destroyed
;   y0 = destroyed
;   y1 = destroyed
;   r4 = destroyed
;   n4 = destroyed
;
; include 'def.asm'
;
; org     phe:
;
; getsync
;   move   #NSYNC,n4           ;number of bits
;   jsr    getvalue           ;get sync right justified
;   move   #>GETSYNCSK,x1     ;mask for sync word hi order
;   and    x1,a               ;mask off any high order 1's
;   rts
```


-190-

```

opt    f=

(c) 1992. Copyright Corporate Computer Systems, Inc. All rights reserved.
\URDCDSYN\getsystd.asm: set led for MPEG-ISO vs old CDQ2000/CDQ1000

title  'Get Syst'

This routine decodes the MUSICAM frame header information.

on exit
x:findidbit      1=high sample rate, 0=low sample rate
y:ctiflgs = updated (PROTECT bit: 0=YES for checksum, 1=NO)
                  (STEREO vs MONO bit: 0=stereo, 1=mono)
                  (JOINT_FRAMING bit: 0=not, 1=joint)
                  (SPLIT_MONO_FRAME bit: 0=no, 1=yes)
x:fnidbit        bit rate code
x:fnidmpl        sampling rate code
y:bitsfrm        actual frame length in bits
x:padbit         0=frame not padded, 1=frame padded w 8 added bits
y:privacybit     privacy bit value in frame header
y:frmtyp         stereo, joint stereo, dual mono or mono
y:sibound        joint stereo intensity boundary subband count
y:maxsubs        number of sub-bands encoded in BAL's
y:copyright      copyright bit value in frame header
y:original       original/home bit value in frame header
y:emphasis       emphasis value in frame header
x:AllwAdd        address of the Allowed table to use
x:skftbl         address of the BAL's bit table to use

a = destroyed
b = destroyed
x0 = destroyed
x1 = destroyed
y0 = destroyed
y1 = destroyed
r0 = destroyed
r1 = destroyed
r4 = destroyed
n4 = destroyed

include 'def.asm'
include 'box_ctl.asm'

org    phe:

```

getsyst:

decode the bits 0 thru 3 of the frame header:

bit description

```

-----
0 - high or low sampling rate:
    1 = high rates 48, 44.1 and 32 K sampling rates
    0 = low rates 24, 22.05 and 16 K sampling rates
1-2 MUSICAM Layer:
    11 = Layer I
    10 = Layer II
    01 = Layer III
3 - CRC-16 checksum frame header protection:

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-191-

```

; 0 = checksum protection encoded after frame header
; 1 = NO checksum protection

move    #NSYSTHDR_1.n4      ;get field #1 (bits 0-3 in hdr)
; bit 0 indicates protection checksum
; 0 = yes checksum included
; 1 = no checksum included

jsr      getvalue           ;get data right justified
move     #>MASKSYSTHDR_1,x1 ;mask for getvalue of header field 1
and      x1,a    #NBITRATE.n4 ;mask off high order bits
; & set len of bit rate-bits 4-7 in hdr

bset     #PROTECT.y:<ctrlflgs ;default that CRC protection applies
move     a1,y:<not_appl       ;see if CRC bit set indicating not appl
jclr     #0,y:<not_appl,_gsyst_00 ;hdr shows zero, CRC is included
bclr     #PROTECT.y:<ctrlflgs ;set that CRC protection NOT applicable

```

_gsyst_00

; set the high or low sampling rate ID code

```

bset     #0,x:fndidbit      ;default with high sample rate bit on
jset     #3,y:<not_appl,_gsyst_01 ;if set for high, continue
bclr     #0,x:fndidbit      ;reset to low sample rate bit on

```

_gsyst_01

; decode the bits 4 thru 7 of the frame header: bit rate

```

jsr      getvalue           ;get bit rate code right justified
move     #>MASKNBITRATE,x1  ;mask for getvalue of frame bit rate
and      x1,a    y:splitrte,x1 ;mask off high order bits
; & get the 1/2 bit rate code
move     a1,x:fndidbit      ;save header bit rate code

```

; test for CDQ2000 split mode of transmission and check for a split mono frame

```

bclr     #SPLIT_MONO_FRAME.y:<ctrlflgs ;clear indication for split mono
jclr     #SPLIT_MODE.y:<ctrlflgs,_gsyst_05 ;test for split mode of trans

move     a1,a               ;clean up junk after getvalue
cmp      x1,a               ;see if frame rate same as split rate
jne      _gsyst_05          ;if not, we should have a full frame

```

; since we matched bit rates, this must be a 1/2 bit rate in mono

```

bset     #SPLIT_MONO_FRAME.y:<ctrlflgs ;indicate for ancillary data

```

_gsyst_05

; decode the bits 8 and 9 of the frame header: sampling rate

```

move     #NSAMPLERATE.n4    ;eat sampling rate
jsr      getvalue           ;get sampling rate right justified
move     #>MASKNSAMPLERATE,x1 ;mask for getvalue of data sampling rate
and      x1,a    #NSYSTHDR_2.n4 ;mask off high order bits
; & set len field #2 (bits 10-11 in hdr)
move     a1,x:fndsmpl       ;save the header sample rate

```

; decode the bits 10 and 11 of the frame header:

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-192-

```

; bit description
; -----
; 10 padding bit:
;     0 = frame is not padded
;     1 = frame is padded with 8 bits
; 11 privacy bit
;
; test the frame padded flag in header (bit 10) and update frame bit count

jsr    getvalue                ;get data right justified
move   #>MASKSYSTHDR_2,x1
and    x1,a    #>PAD_SLOT,x1  ;mask off high order bits
; & get the padded bits added to frame
move   a1,y:<not_appl         ;see if frame padded bit set
move   y:frmbits,a           ;get the unpadded frame bit count
bclr   #0,x:padbit            ;default that the frame is not padded
jclr   #1,y:<not_appl,_gsyst_06 ;if hdr bit not set, no padded bits
bset   #0,x:padbit            ;indicate padded bits
add    x1,a                  ;add pad bits to frame bit count

_gsyst_06

;set the frame length in bits (normal or padded with 8 bits)
;set the frame privacy bit in header (bit 11)

move   a,y:bitsfrm            ;store actual frame bit count
bclr   #0,y:privacybit        ;default the frame header privacy bit
CLR_PRIVACY_BIT_DCD           ;in decoder status
jclr   #0,y:<not_appl,_gsyst_08
bset   #0,y:privacybit        ;set the frame header privacy bit
SET_PRIVACY_BIT_DCD           ;in decoder status

_gsyst_08

;decode the bits 12 and 13 of the frame header: frame type
; 00 = FULL STEREO    (2 channels)
; 01 = JOINT STEREO   (2 channels)
; 10 = DUAL MONO      (2 channels)
; 11 = MONO           (1 channel)

move   #NFRAMEYPE,n4          ;get frame type (bits 12-13 in hdr)
jsr    getvalue                ;get frame type right justified
move   #>MASKFRAMEYPE,x1       ;mask for getvalue of framing type
and    x1,a    #NSTINTENSITY,n4 ;mask off high order bits
; & get stereo intensity (bits 14-15)
move   a1,y:frmtpe            ;save type of frame

;set the default MAXSUBBANDS as for 2 channel frames

move   #oldccs,r0              ;to test if old CCS CDQ frames
move   y:maxsubs_2,y1          ;default to 2 channel MAXSUBBANDS

; if the old CCS flag is set to decode from old CCS CDQ's, use mono MAXSUBBANDS

jclr   #0,y:(r0),_gsyst_09     ;if MPEG-ISO, continue
move   y:maxsubs_1,y1          ;default to MONO MAXSUBBANDS

_gsyst_09

;set the type of frame flag

```

SUBSTITUTE SHEET (RULE 26)

-193-

```

move    y:frmttype,a           ;get the frame type
move    #>FULL_STEREO,x1
cmp     x1,a    #>JOINT_STEREO,x1
jne     _gsyst_10
bclr    #STEREO vs MONO,y:<ctlflgs    ;indicate stereo samples
bclr    #JOINT_FRAMING,y:<ctlflgs    ;clear joint stereo indicator
jmp     _gsyst_40

_gsyst_10
cmp     x1,a    #>DUAL,x1
jne     _gsyst_20
bclr    #STEREO vs MONO,y:<ctlflgs    ;indicate stereo samples
bset    #JOINT_FRAMING,y:<ctlflgs    ;indicate stereo samples
jmp     _gsyst_40

_gsyst_20
cmp     x1,a
jne     _gsyst_30
bclr    #STEREO vs MONO,y:<ctlflgs    ;dual channel is same as stereo
bclr    #JOINT_FRAMING,y:<ctlflgs    ;indicate stereo samples
bclr    #JOINT_FRAMING,y:<ctlflgs    ;clear joint stereo indicator
jmp     _gsyst_40

_gsyst_30
bset    #STEREO vs MONO,y:<ctlflgs    ;indicate mono samples
bclr    #JOINT_FRAMING,y:<ctlflgs    ;clear joint stereo indicator

;set the MAXSUBBANDS for MONO channel frames

move    y:maxsubs_1,y1           ;get to MONO MAXSUBBANDS

;if SPLIT_MONO_FRAME, use split frame mono MAXSUBBANDS

jclr    #SPLIT_MONO_FRAME,y:<ctlflgs,_gsyst_40
move    y:splmaxsubs,y1           ;get to split MONO MAXSUBBANDS

_gsyst_40

;set the number of sub-bands encoded in the BAL's

move    y1,y:<maxsubs             ;set the working MAXSUBBANDS for frame

; light led to indicate MPEG-ISO compatible frames
; or old CCS CDQ2000/CDQ1000 non-conforming frames at low bit rates

move    #oldccs,r0               ;to test if old CCS CDQ frames
nop
jclr    #0,y:(r0),_iso_led        ;if ISO, set led as ISO
ON MPEG_ISO vs CCS_LED_DCD        ;indicate old ccs frames
jset    #1,y:(r0),do_leds         ;if CDQ1000, set led as CCS
jset    #STEREO vs MONO,y:<ctlflgs,_iso_led ;if MONO, ISO led
move    #>SAM48K,x0               ;test for 48 K sampling
move    #>SAM32K,x1               ;test for 32 K sampling
move    #>BITRATE_56,y0           ;low bit rate code 56 K
move    y:smplrte,a               ;to test sample rate code
cmp     x0,a    #>BITRATE_96,y1    ;see if 48 K sampling
; & set hi bit rate 96 K @ 48
jeq     _tst_bit                  ;if 48, test bit rate range
cmp     x1,a    #>BITRATE_160,y1   ;see if 32 K sampling
; & set hi bit rate 96 K @ 32

```

SUBSTITUTE SHEET (RULE 26)

-194-

```

jne    _iso_led    ;if not 32, set ISO led

_tst_bit
move   y:bitrate,a    ;check bit rate in the range
cmp    y0,a           ;test vs lowest ISO high code
jlt    _iso_led       ;if less, ISO led
cmp    y1,a           ;test vs highest ISO high code
jle    _do_leds       ;if less or equal, leave CCS led

_iso_led
OFF_MPEG_ISO_vs_CCS_LED_DCD    ;indicate iso compatible frames

_do_leds
SET_LEDS_DCD

;decode the bits 14 and 15 of the frame header:
; mode extension (joint stereo intensity boundary)
; 00 = stereo for sub-bands 0 thru 3, joint for sub-bands 4 and up
; 01 = stereo for sub-bands 0 thru 7, joint for sub-bands 8 and up
; 10 = stereo for sub-bands 0 thru 11, joint for sub-bands 12 and up
; 11 = stereo for sub-bands 0 thru 15, joint for sub-bands 16 and up

jsr    getvalue      ;get data right justified
move   #>MASKSTINTENSITY,x1    ;mask for getvalue of intensity bound
and    x1,a    #BOUND_4,r0    ;mask off high order bits
; & set up for joint just in case
jclr   #JOINT_FRAMING,y:<ctlflgs,_gsyst_90 ;intensity is meaningless

move   a1,a          ;clear off any junk
move   #>INTENSITY_4,b    ;get code for channels 4-31 intensity
cmp    a,b    #>INTENSITY_8,b
jeq    _gsyst_90

cmp    a,b    #>INTENSITY_12,b
jne    _gsyst_80      ; not joint, intensity is meaningless
move   #BOUND_8,r0
jmp    _gsyst_90

_gsyst_80
cmp    a,b    #BOUND_16,r0
jne    _gsyst_90      ; not joint, intensity is meaningless
move   #BOUND_12,r0

_gsyst_90
move   r0,y:sibound    ;save intensity stereo sub-band bound

;decode the bits 16 thru 19 of the frame header:
; bit description
; -----
; 16 copyright bit:
;     0 = no copyright
;     1 = protected by copyright
; 17 original/home bit:
;     0 = bitstream is a copy
;     1 = bitstream is an original
; 18-19 emphasis:
;     00 = no emphasis
;     01 = 50/15 microsec. emphasis
;     10 = reserved

```

SUBSTITUTE SHEET (RULE 26)

-195-

```

11 = CCITT J.17 emphasis

move    #NSYSTHDR_3,n4      ;get field #3 (bits 16-19)
jsr      getvalue           ;get data right justified

move     #>MASKSYSTHDR_3,x1 ;to mask off unwanted bits
and      x1,a               ;mask off the unwanted bits
move     a1,y:<not_appl      ;move to addr to be tested
clr      a                  ;to restore y:<not_appl as all 0's

;set the copyright bit, original/home bit and emphasis code from header

bclr     #0,y:copyright      ;default bit as not set
CLR_COPYRIGHT_BIT_DCD        ;in decoder status
jclr     #3,y:<not_appl,_gsyst_91 ;if bit 16 not set, continue
bset     #0,y:copyright      ;set the copyright bit
SET_COPYRIGHT_BIT_DCD        ;in decoder status

_gsyst_91
bclr     #0,y:original        ;default bit as not set
CLR_ORIGINAL_BIT_DCD         ;in decoder status
jclr     #2,y:<not_appl,_gsyst_92 ;if bit 17 not set, continue
bset     #0,y:original        ;set the original/home bit
SET_ORIGINAL_BIT_DCD         ;in decoder status

_gsyst_92
move     a,y:emphasis         ;zero the emphasis code
CLR_EMPHASIS_BIT_0_DCD       ;in decoder status
CLR_EMPHASIS_BIT_1_DCD       ;in decoder status
jclr     #1,y:<not_appl,_gsyst_93 ;if bit 18 not set, try bit 19
bset     #1,y:emphasis        ;set bit 1 of emphasis code
SET_EMPHASIS_BIT_1_DCD       ;in decoder status

_gsyst_93
jclr     #0,y:<not_appl,_gsyst_94 ;if bit 19 not set, finish up
bset     #0,y:emphasis        ;set bit 0 of emphasis code
SET_EMPHASIS_BIT_0_DCD       ;in decoder status

_gsyst_94
;restore y:<not_appl to all zeros

move     a,y:<not_appl        ;reset the dummy variable

;Set the proper Allowed table and BAL's bit table addresses:
;test for low sampling rate Allowed table

move     #smplidbit,r0        ;addr of frame header ID bit (0 = low)
nop
jset     #0,y:(r0),_gsyst_95   ;if high rate, select Allowed table

move     #Allowed_3,r0        ;addr of low sampling allowed table
move     #skftbl_3,r1         ;addr of low sampling BAL's bit table
jmp      _gsyst_100           ;go to store Allowed table address

_gsyst_95
;Set the proper Allowed table address based on working MAXSUBBANDS (y:<maxubs)
; if less than 27, used table 2

```

SUBSTITUTE SHEET (RULE 26)

-196-

```

move    y:<maxsubs,x0      ;get current MAXSUBBANDS
move    #>27,a             ;to see which of 2 tables applies
move    #skftbl_1,r1       ;addr of high sampling BAL's bit table
cmp     x0,a    #Allowed_1,r0 ;see if need the low bit rate table
                                ; & set up as regular Allowed table
jle     _gsyst_100         ;regular Allowed table applies

;select the lower bit rate Allowed table

move    #Allowed_2,r0
move    #skftbl_2,r1       ;addr of high sampling BAL's bit table

_gsyst_100

;set the address of the selected Allowed table
;set the address of the selected BAL's bit table

move    r0,x:AllwAdd
move    r1,x:skftbl

rts

```

-197-

```

opt      fc
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.
; \URDCDSYN\synth.asm

title 'Synthesize a group of sample and output audio'

;synth.asm: this is the main of the poly synthesis routine
; it handles a new group of samples to be decoded and inverse quantized
; for stereo a group of samples contains 192 samples (96 left & 96 right)
; if mono a group of samples contains 96 samples only

include 'def.asm'
include 'box_ctl.asm'

section highmisc
xdef    dualchan
xdef    synthN6Save

org     yhe:
stsynth_yhe

dualchan    ds    1           ;control for channel swap ctls
synthN6Save ds    1           ;instead of ssh
                                ;bit 0 = 1 means copy left to right
                                ;bit 1 = 1 means copy right to left
                                ;bit 2 = 1 means swap left & right
                                ;bit 3 = 1 means mute both left & right

endsynth_yhe
endsec

org     phe:

synth
move    #dualchan,r0           ;set addr of two chan ctls
move    #ASMDData,r1          ;position to left channel

;see if the frame is to be muted
jclr    #MUTE_LEFT_and_RIGHT,y:(r0),_synt_00

;set the number of words in both channels for the MUTE do loop

move    #NUMSUBBANDS*NPERGROUP*2,n0 ;2 channels numb words to mute
move    #0,n1                       ;hold position at left channel
jmp     _synt_20                     ;go to the mute loop

_synt_00

;if a stereo frame, checkout for special mute or swaps
jclr    #STEREO_vs_MONO,y:<ctlflgs,_synt_40

move    #NUMSUBBANDS*NPERGROUP,n1   ;spacing to right channel
move    r1,r0                       ;position to left channel
move    (r1)+n1                     ;addr of right channel

;copy the left into right

```



-198-

```

do      #NUMSUBBANDS*NPERGROUP,_synt_05
move    x:(r0)+,x0
move    x0,x:(r1)+      ;get left channel value
                        ;put left value into right
_synt_05
; if we do not have to mute a channel (mono to both),
; skip ahead to restore registers used
jset    #MONO_OUT_BOTH,y:<ctlflgs,_synt_90 ;out to both, go restore regs
; set the number of words in one channel for the mute do loop
move    #NUMSUBBANDS*NPERGROUP,n0      ;1 channel numb words to mute
; set up to mute the channel not selected for mono output
move    #ASMDData,r1
move    #0,n1      ;position to left channel
                        ;start at left channel
; if not the left channel for output, continue
; else, position to the right channel for muting
jset    #MONO_OUT_CHANNEL,y:<ctlflgs,_synt_20 ;if right, zero left
move    #NUMSUBBANDS*NPERGROUP,n1      ;else, zero the right channel
_synt_20
; mute the proper channel(s)
move    #0,x0
move    (r1)+n1      ;to mute the channel
                        ;addr of channel to mute
do      n0,_synt_30
move    x0,x:(r1)+      ;zero value in chosen channel
_synt_30
jmp     _synt_90      ;do the polysynthesis
_synt_40
; see if the two channel frame requires any swapping:
; swap left and right
; left into right
; right into left
jclr    #SWAP_LEFT_and_RIGHT,y:(r0),_synt_50
; swap the left and right channels
move    #NUMSUBBANDS*NPERGROUP,n1      ;spacing to right channel
move    r1,r0      ;position to left channel
move    (r1)+n1      ;addr of right channel
; copy the left into right
do      #NUMSUBBANDS*NPERGROUP,_synt_45
move    x:(r0),x0      ;get left channel value
move    x:(r1),x1      ;get right channel value

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-199-

```

        move    x0,x:(r1)+      ;put left value into right
        move    x1,x:(r0)+      ;put right value into left

_synt_45
        jmp     _synt_80        ;go see if any channel mutes

_synt_50
;see if a copy the left into the right
        jclr    #COPY_LEFT_to_RIGHT,y:(r0),_synt_60 ;if not copy left to right
;copy the left channel into the right channel
        move    #NUMSUBBANDS*NPARGROUP,n1      ;spacing to right channel
        move    r1,r0                          ;position to left channel
        move    (r1)+n1                        ;addr of right channel
        jmp     _synt_70                       ;do the copy

_synt_60
;see if a copy the right into the left
        jclr    #COPY_RIGHT_to_LEFT,y:(r0),_synt_80 ;if not copy right to left
;copy the right channel into the left channel
        move    #NUMSUBBANDS*NPARGROUP,n0      ;spacing to right channel
        move    r1,r0                          ;position to left channel
        nop
        move    (r0)+n0                        ;addr of right channel

_synt_70
;copy the one channel into the other
        dc      #NUMSUBBANDS*NPARGROUP,_synt_80
        move    x:(r0)+,x0                    ;get source channel value
        move    x0,x:(r1)+                    ;put source value into destin

_synt_80
;see if either channel is to be muted
        jmp     _synt_05

_synt_90
;pass both channels to the polysynthesis routine
        move    #ASMDData,r0
        move    n6,y:synthN6Save              ;save
        move    #1023,m2                      ;set to be a mod(1024) buffer
        move    m2,m3                         ;set to be a mod(1024) buffer
        move    #32,n0                        ;set scale factor

        jsr     polysynt
        move    y:synthN6Save,n6              ;restore n6

```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



-200-

```
move    y:linear,m1      ;restore to linear addressing
move    m1,m2            ;restore to linear addressing
move    m1,m3            ;restore to linear addressing
move    m1,m5            ;restore to linear addressing

rts
```

-201-

```
; (c) 1991. Copyright Corporate Computer Systems, Inc. All rights reserved.  
; c:\musicam\dsp\acorn\urdcdsyn\translte.asm  
include '..\ultma\translte.asm'
```

SUBSTITUTE SHEET (RULE 26)

BAD ORIGINAL



CLAIMS

What is claimed is:

1. An audio transmission system comprising:
a coder for coding an input audio signal into
5 a digital signal to be transmitted through a
traditional analog telephone network, the digital
signal having a transmission rate of 28.8 kilobits
per second or less; and
a decoder for decoding the digital signal that
10 is received from the telephone network to provide
an output audio signal with a frequency range
greater than 4 kilohertz.

10

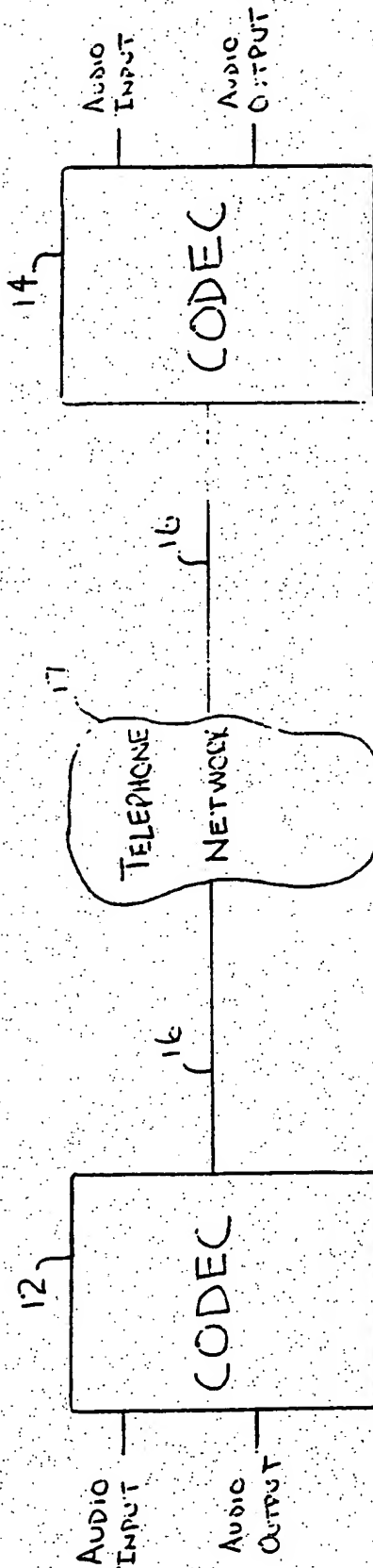


Fig. 1

BAD ORIGINAL

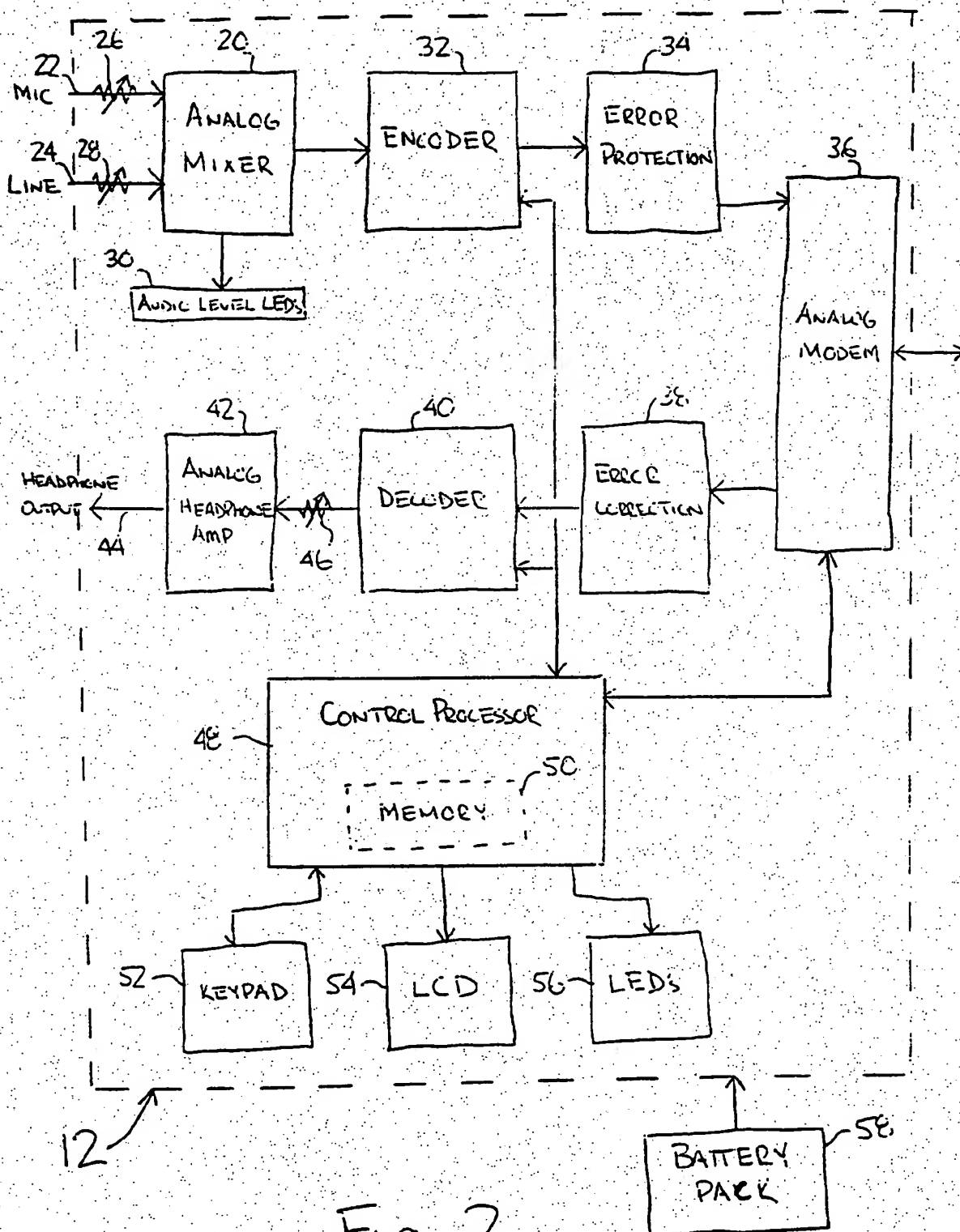


Fig. 2

3/20

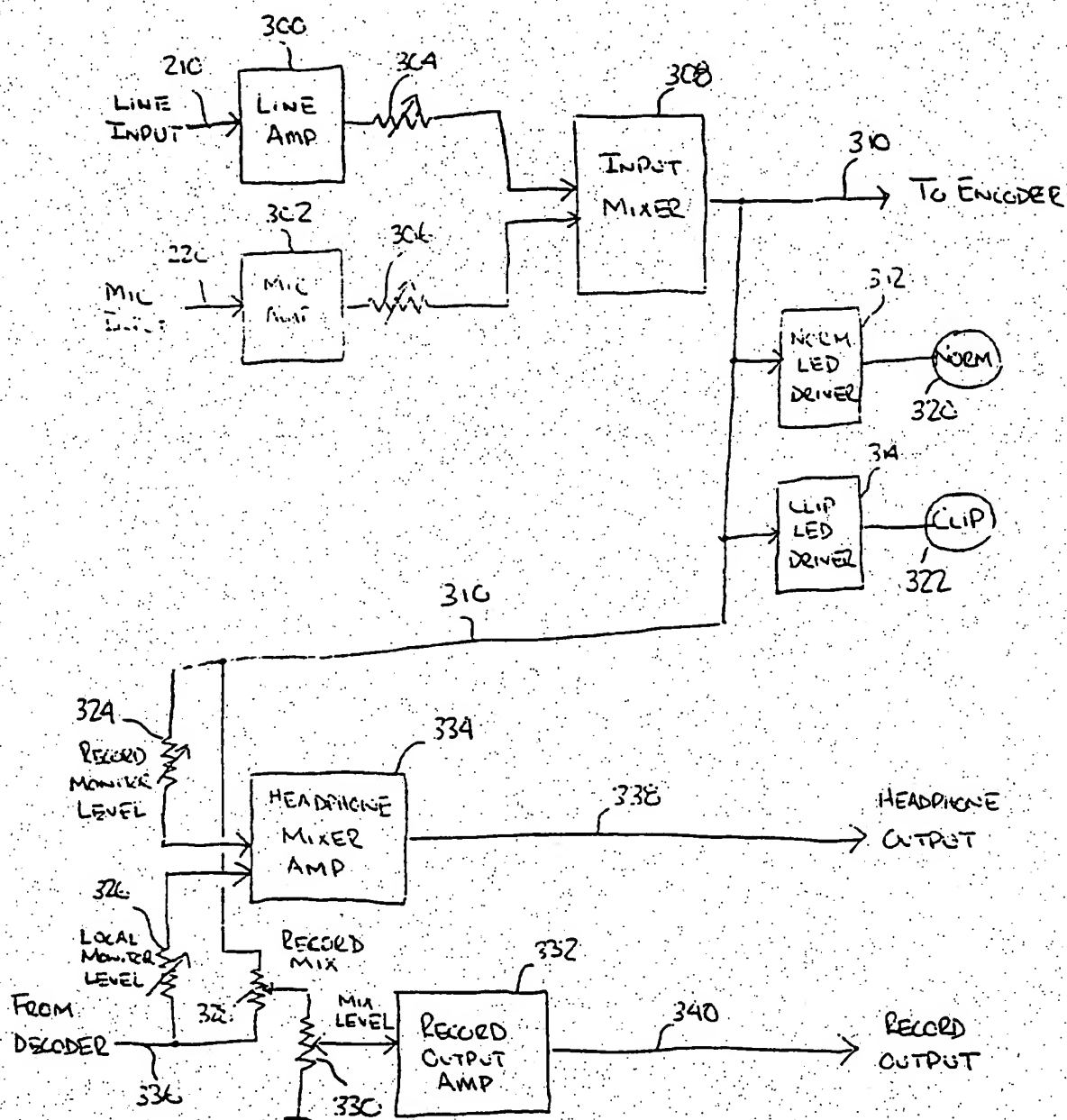
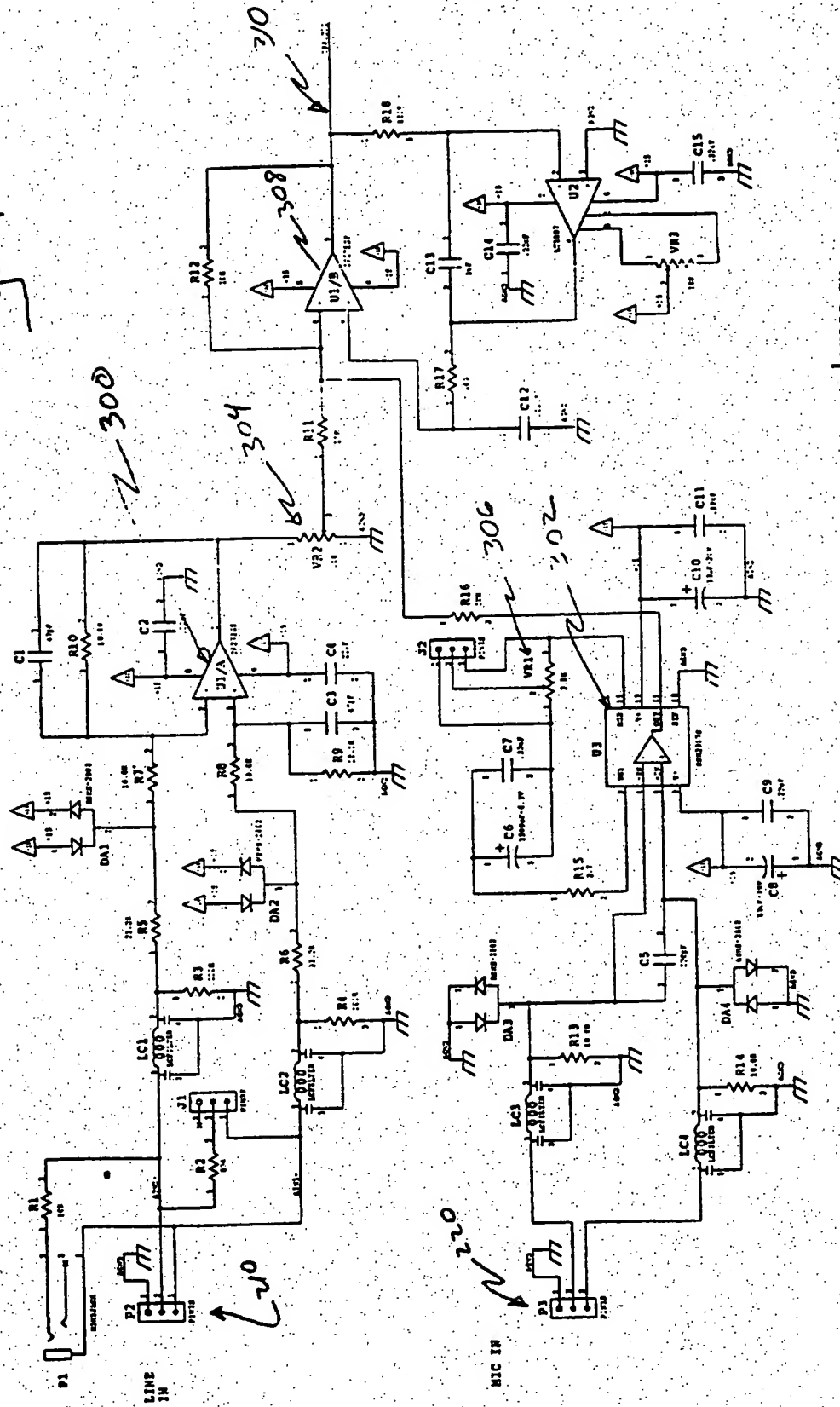
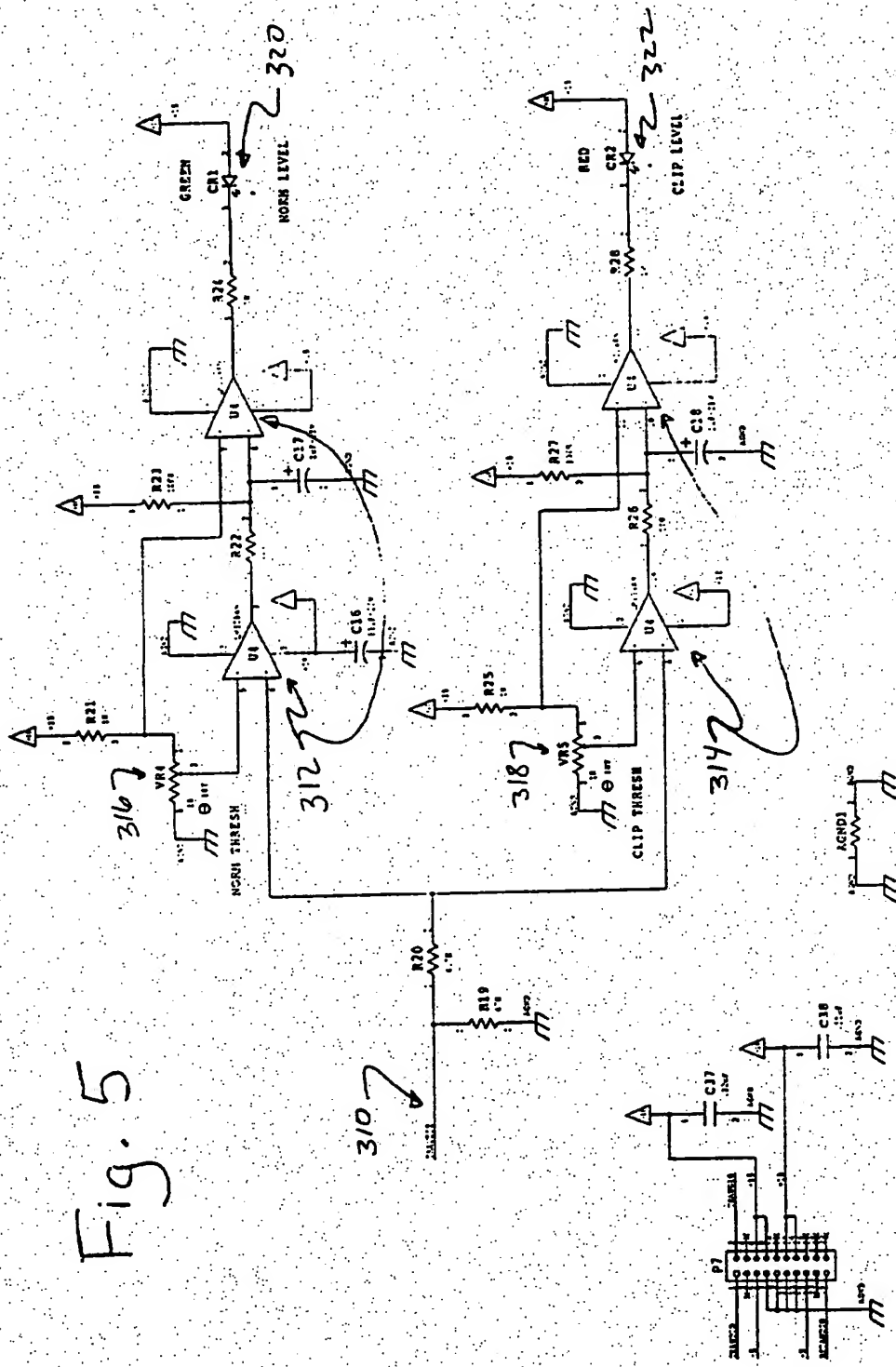


Fig. 3

4/20

Fig. 4



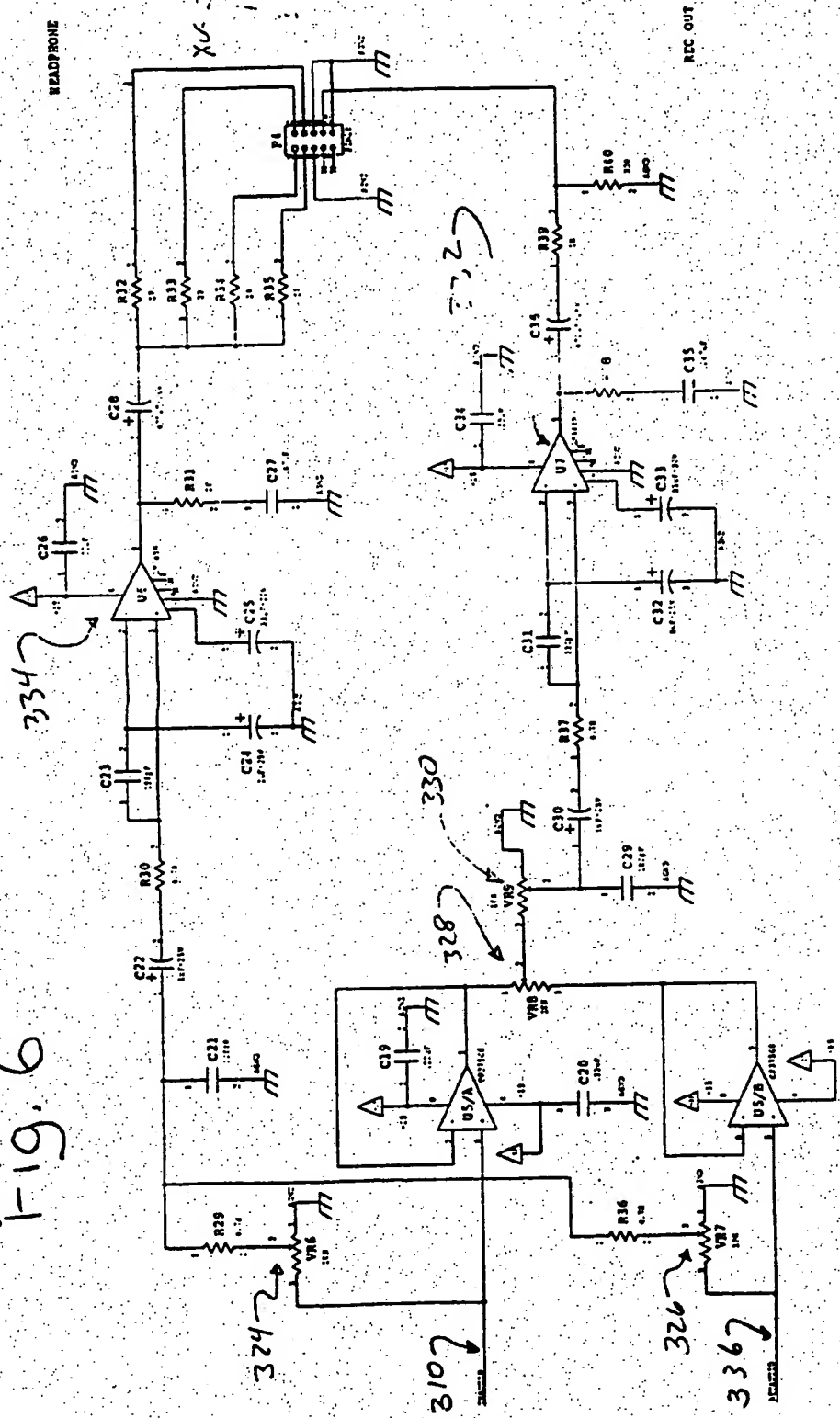


BAD ORIGINAL



6/20

Fig. 6



HEADPHONE AMP

BAD ORIGINAL

7/20

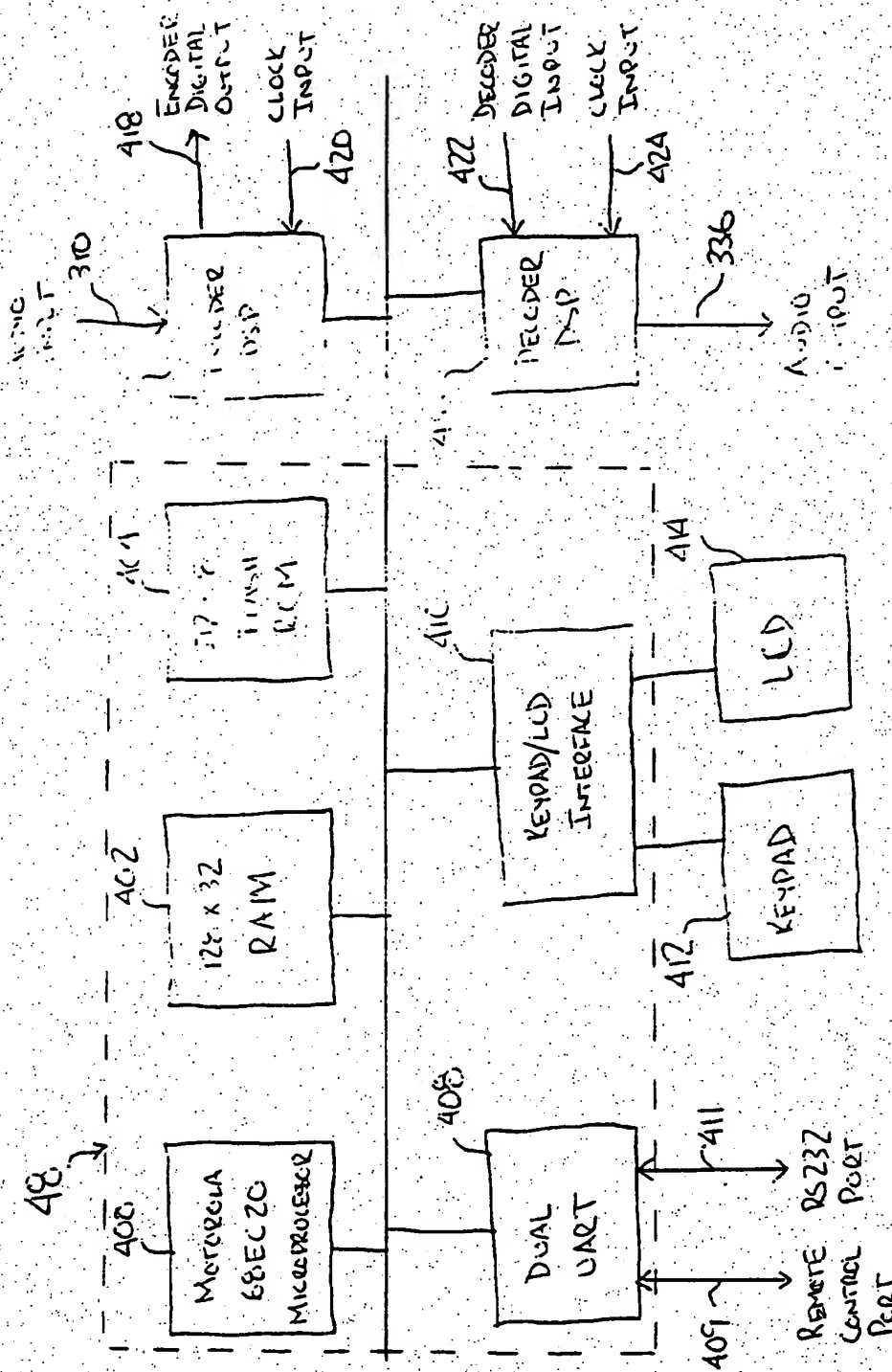


Fig 7

Fig. 1

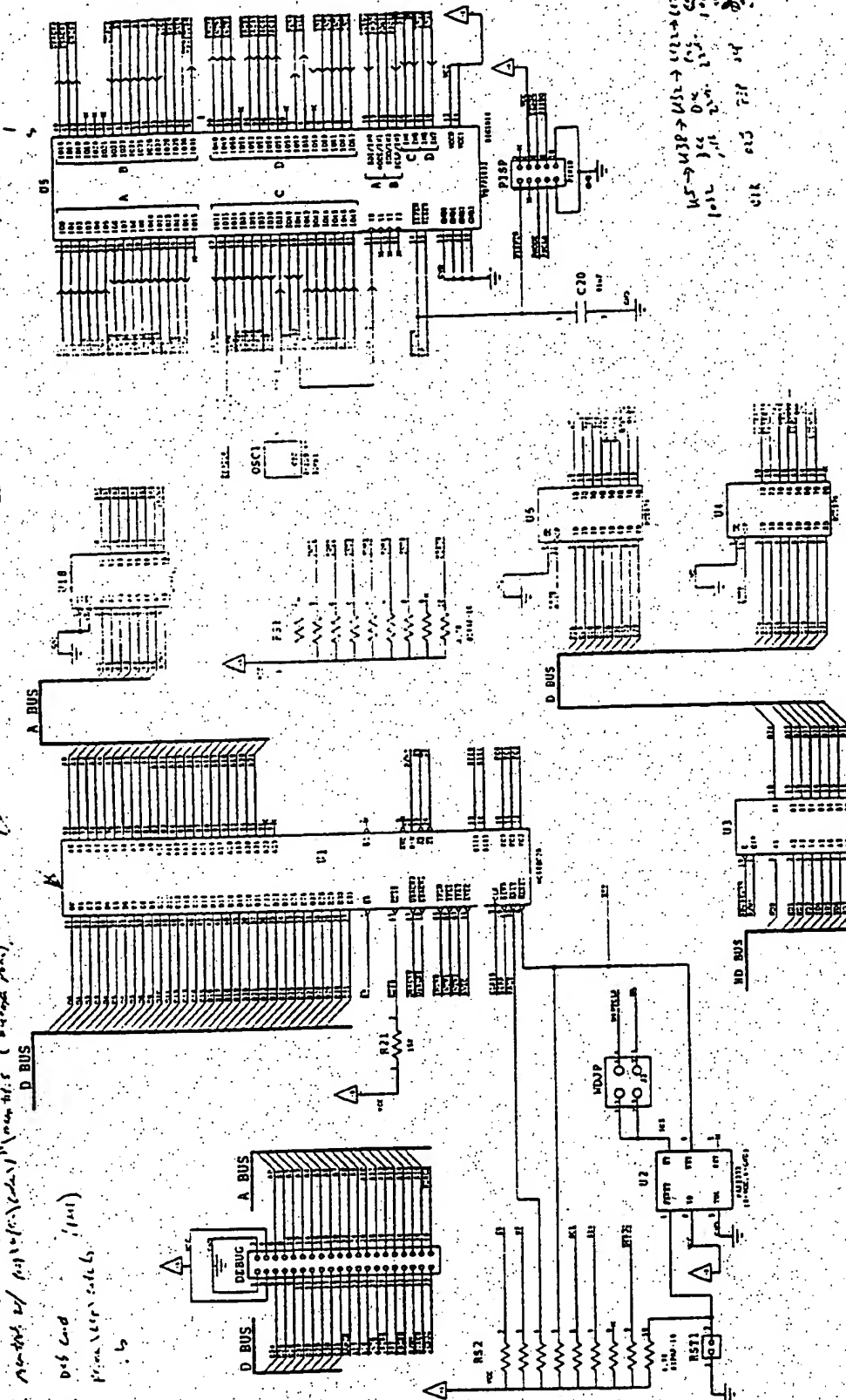
400

1. Monitor w/ bus controller (bus master)

2. Bus card

3. Bus card

4. Bus card

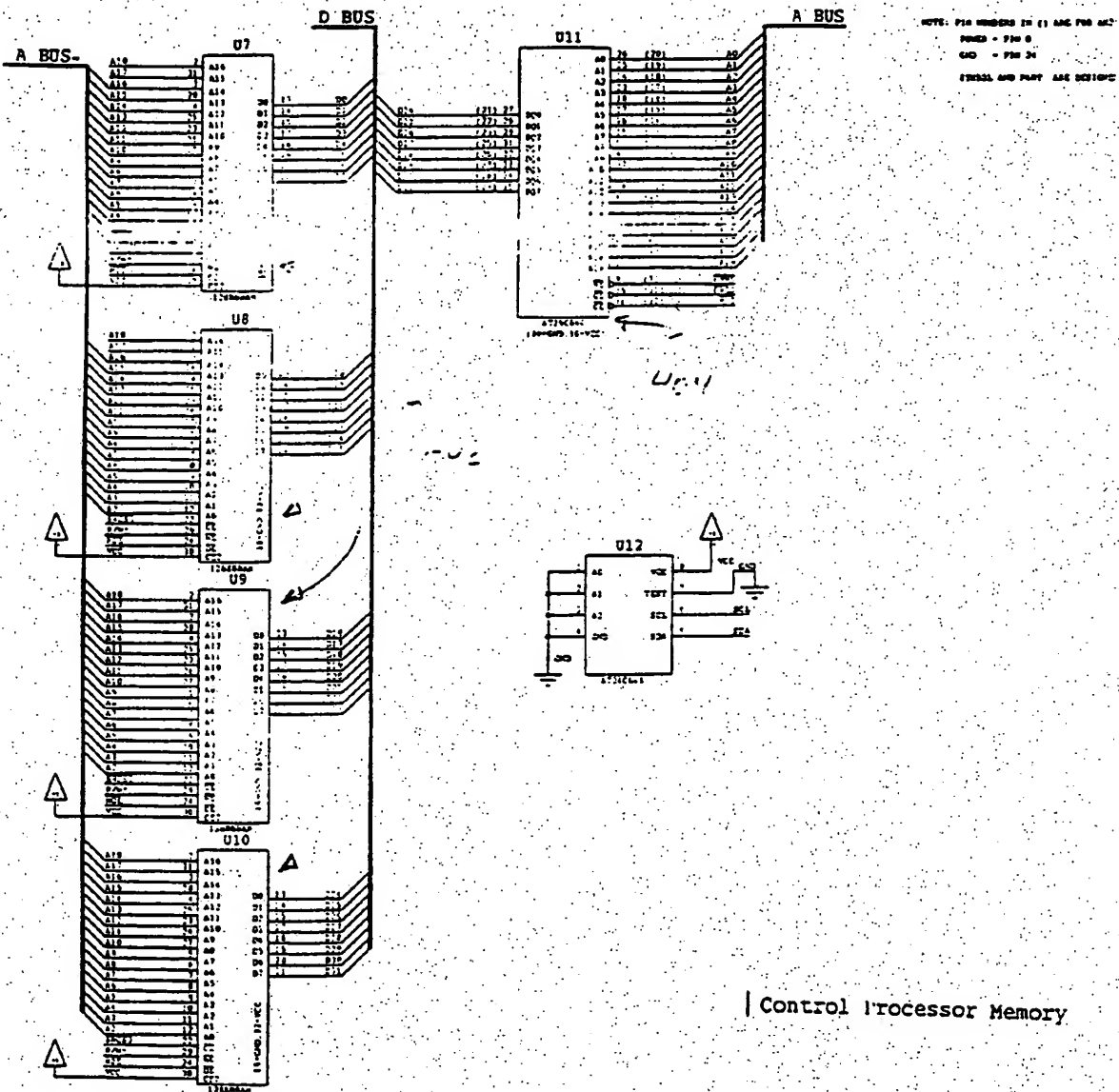


BAD ORIGINAL



9/20

Fig. 9

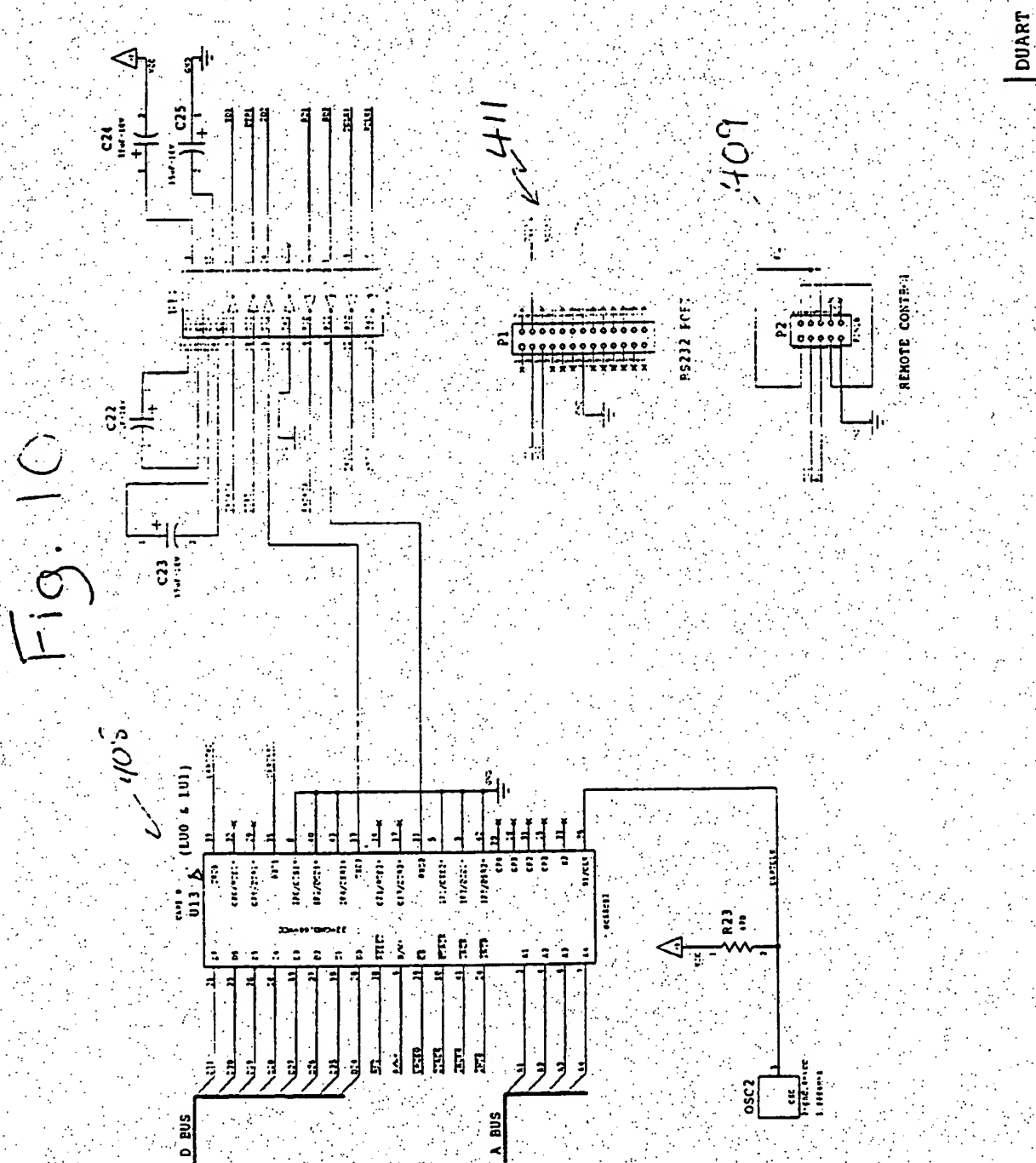


Control Processor Memory

BAD ORIGINAL

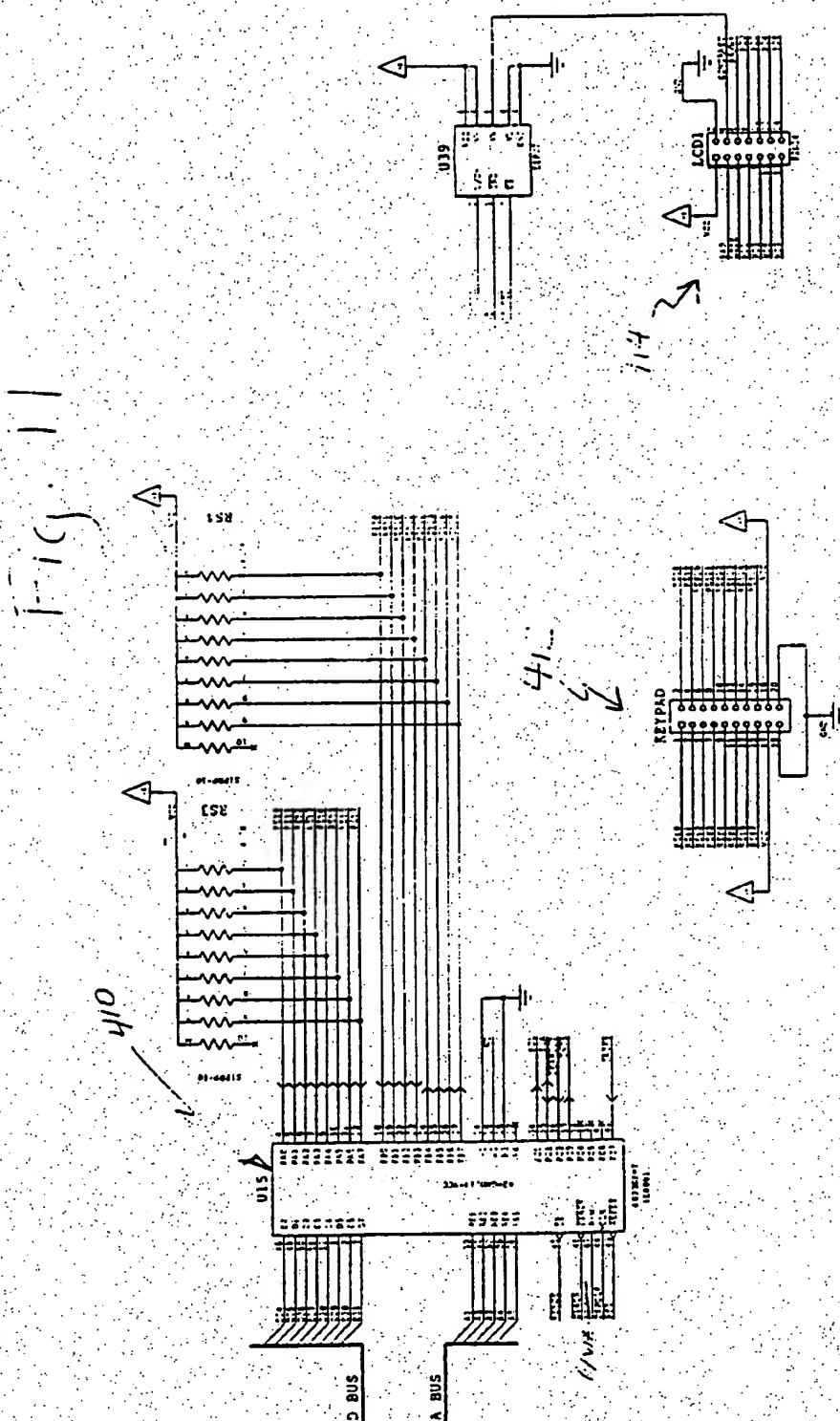


10/20



BAD ORIGINAL

11/20



पृ०

BAD ORIGINAL

12/20

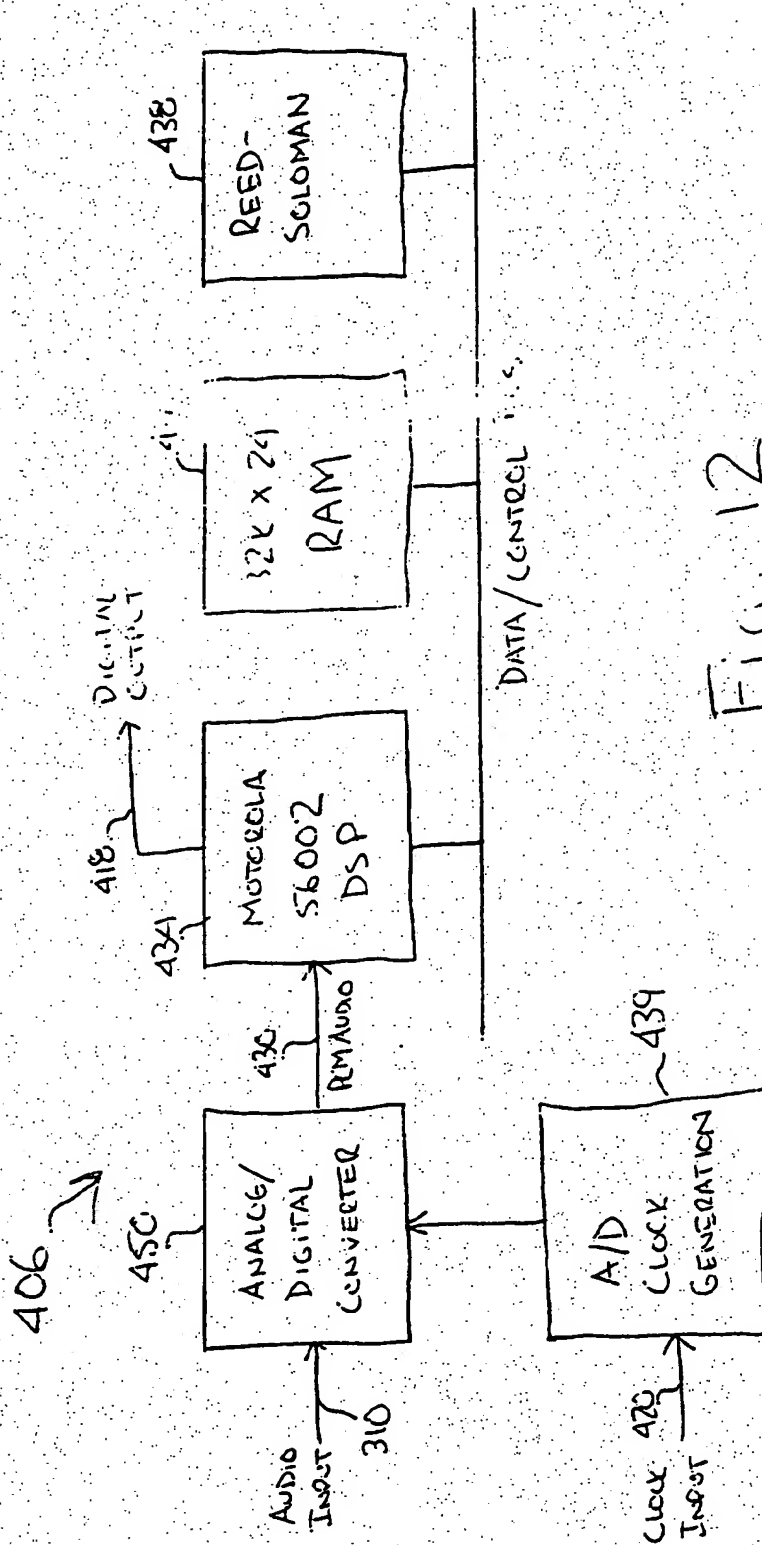
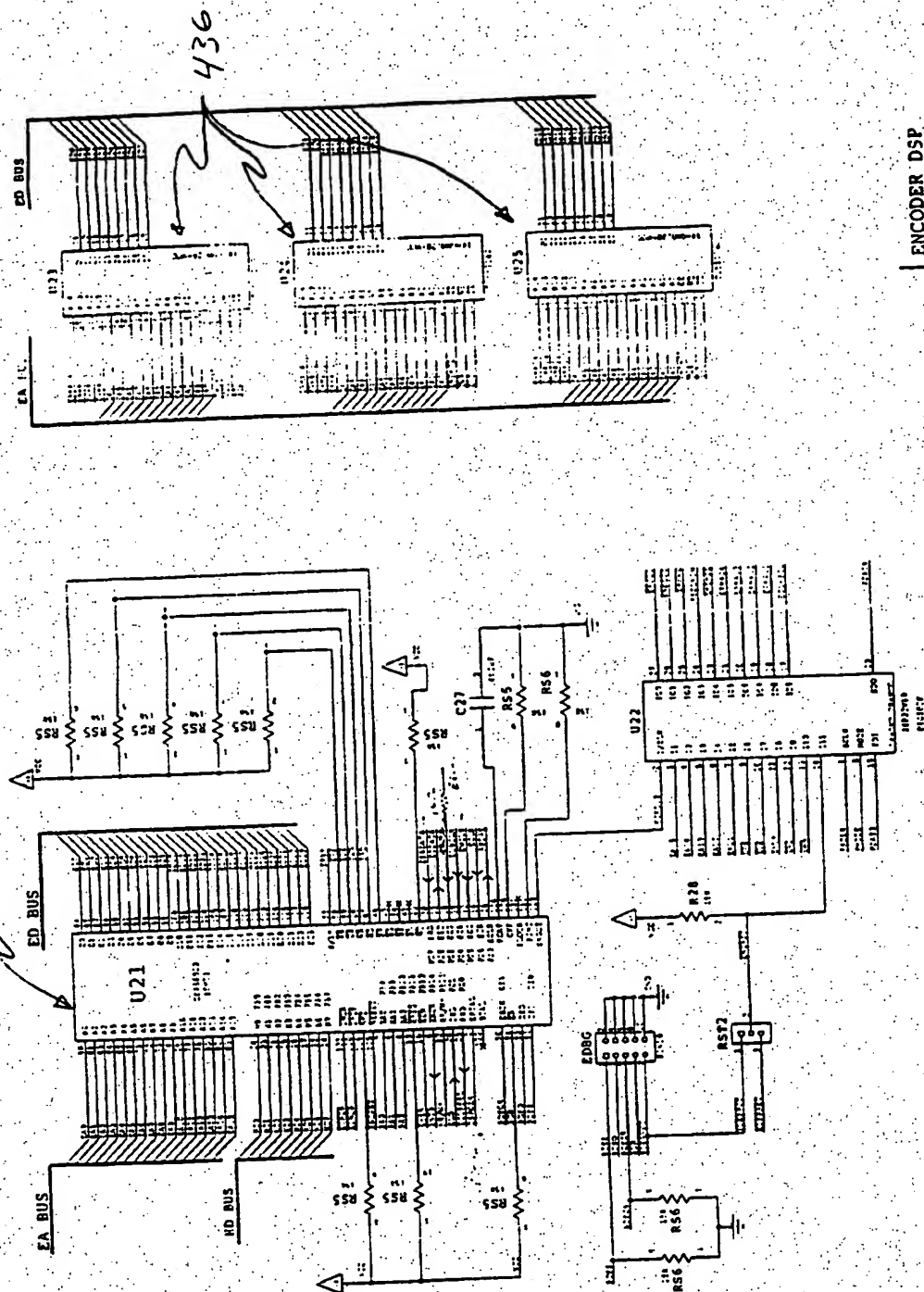


Fig. 12

12/20

BAD ORIGINAL

File 13

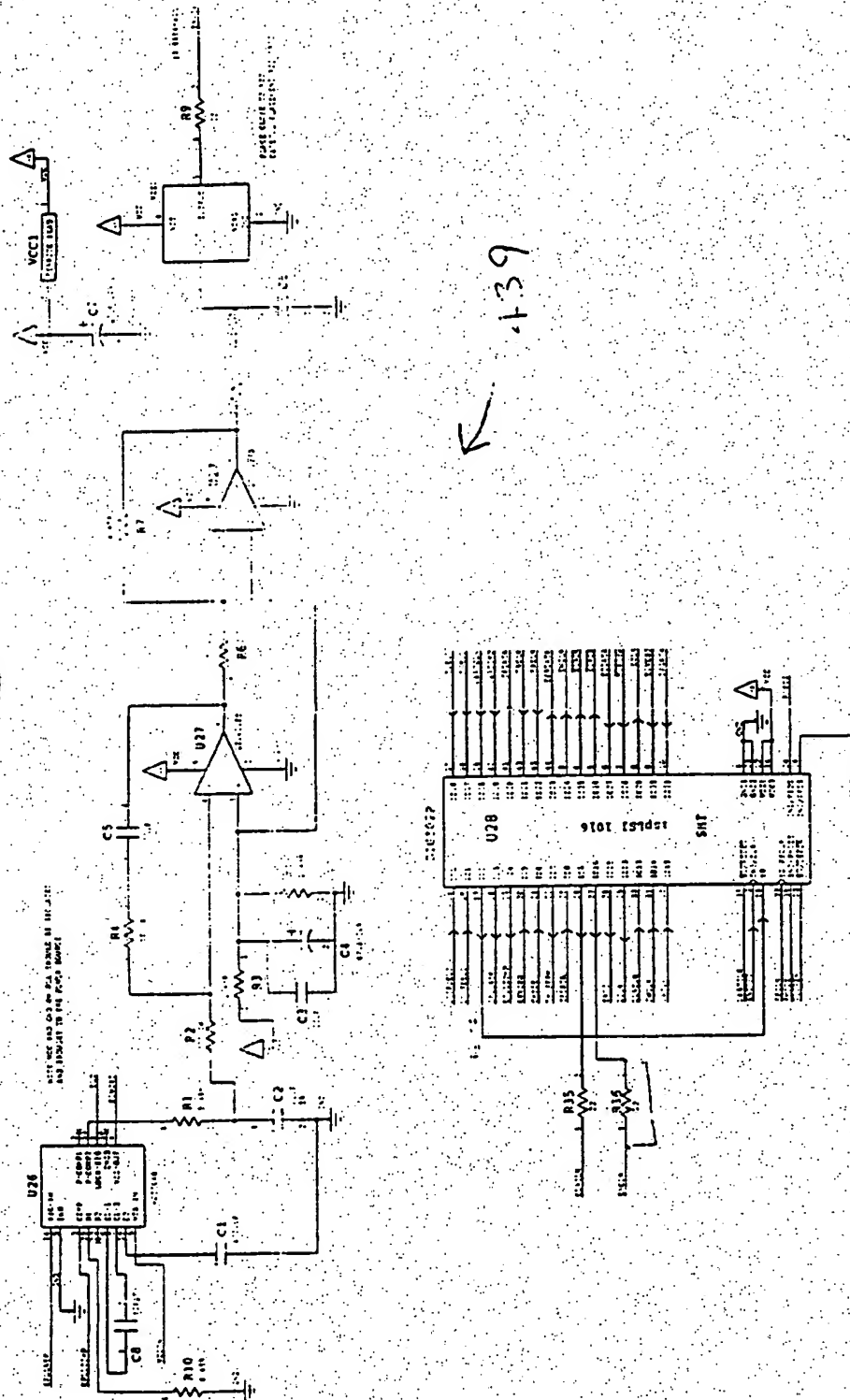


ENCODER DSP

BAD ORIGINAL

14/20

Fig. 14

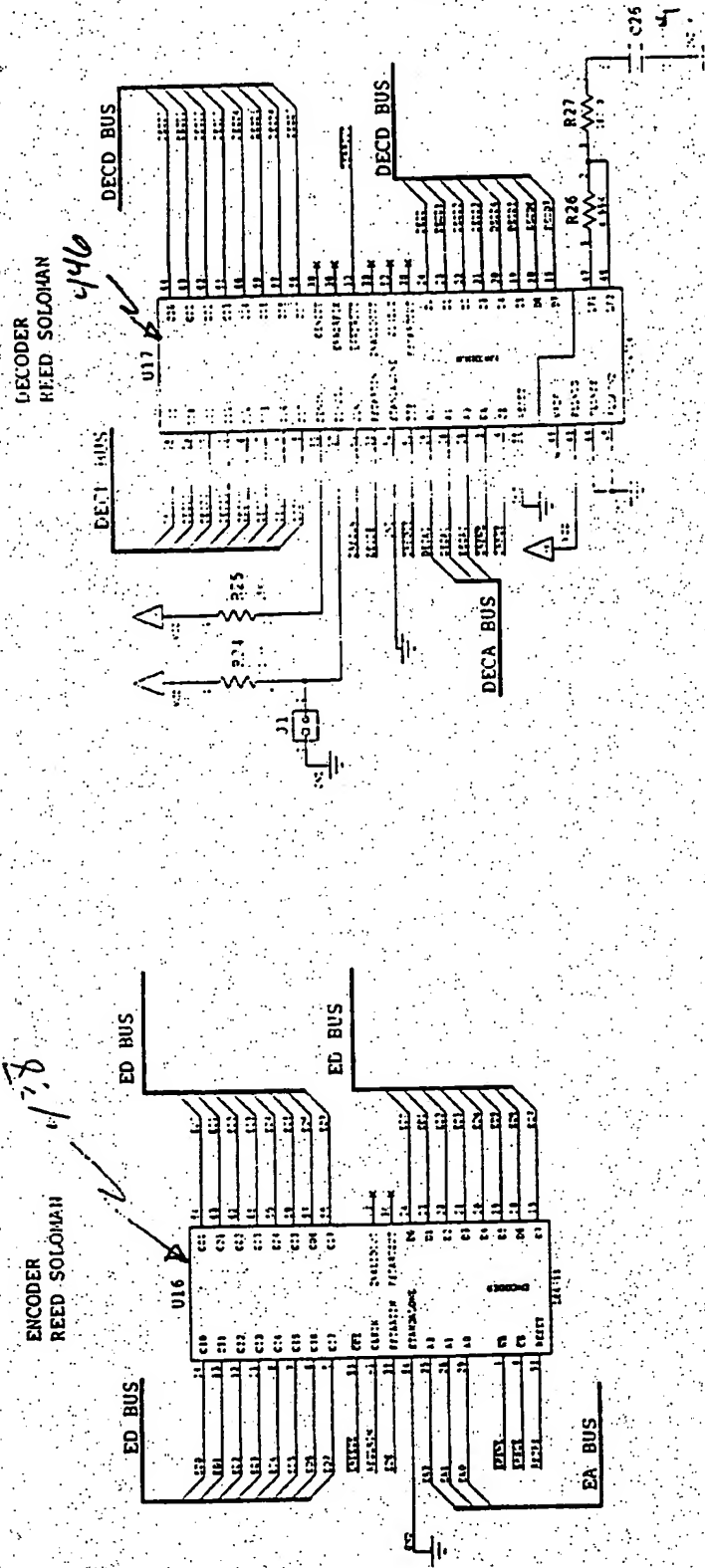


139

ENCODER PLL

BAD ORIGINAL

Fig. 15



REED-SOLOMAN

BAD ORIGINAL

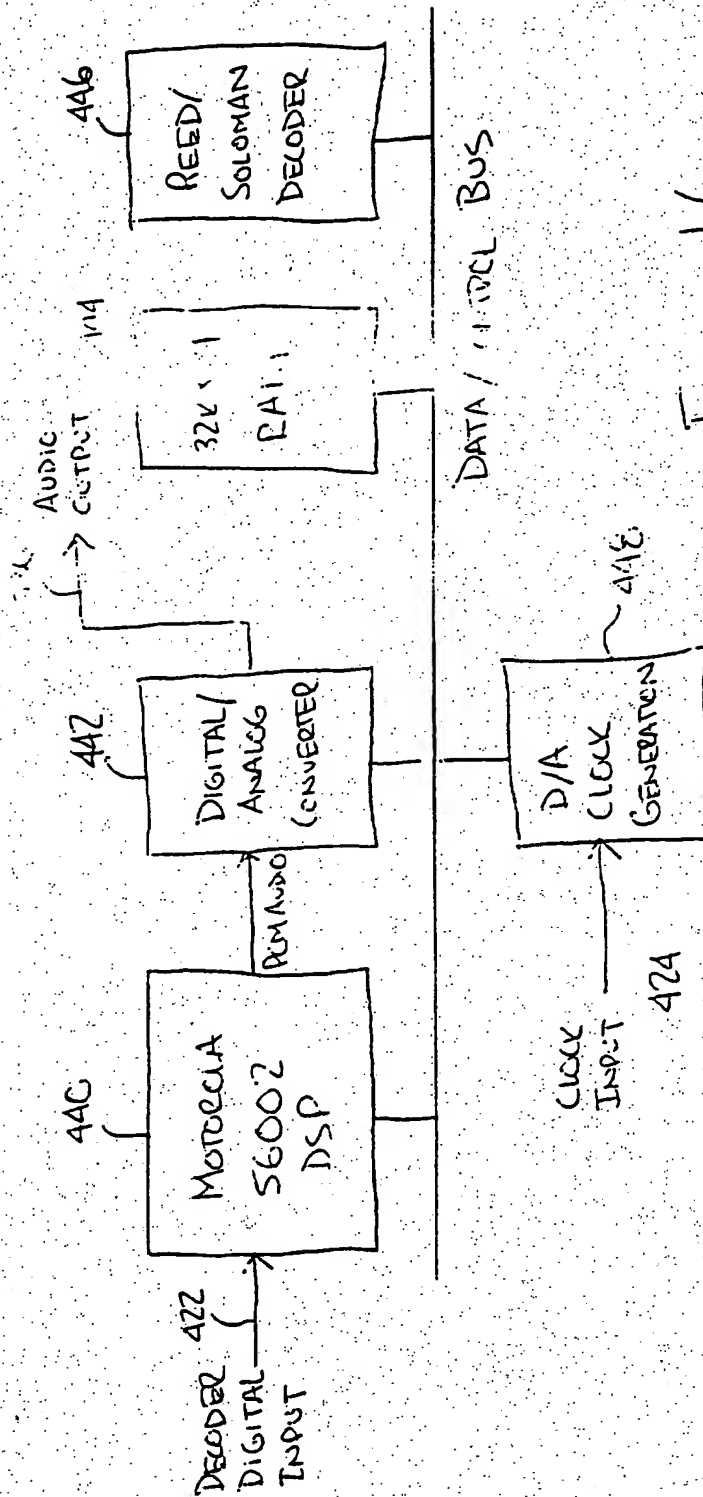
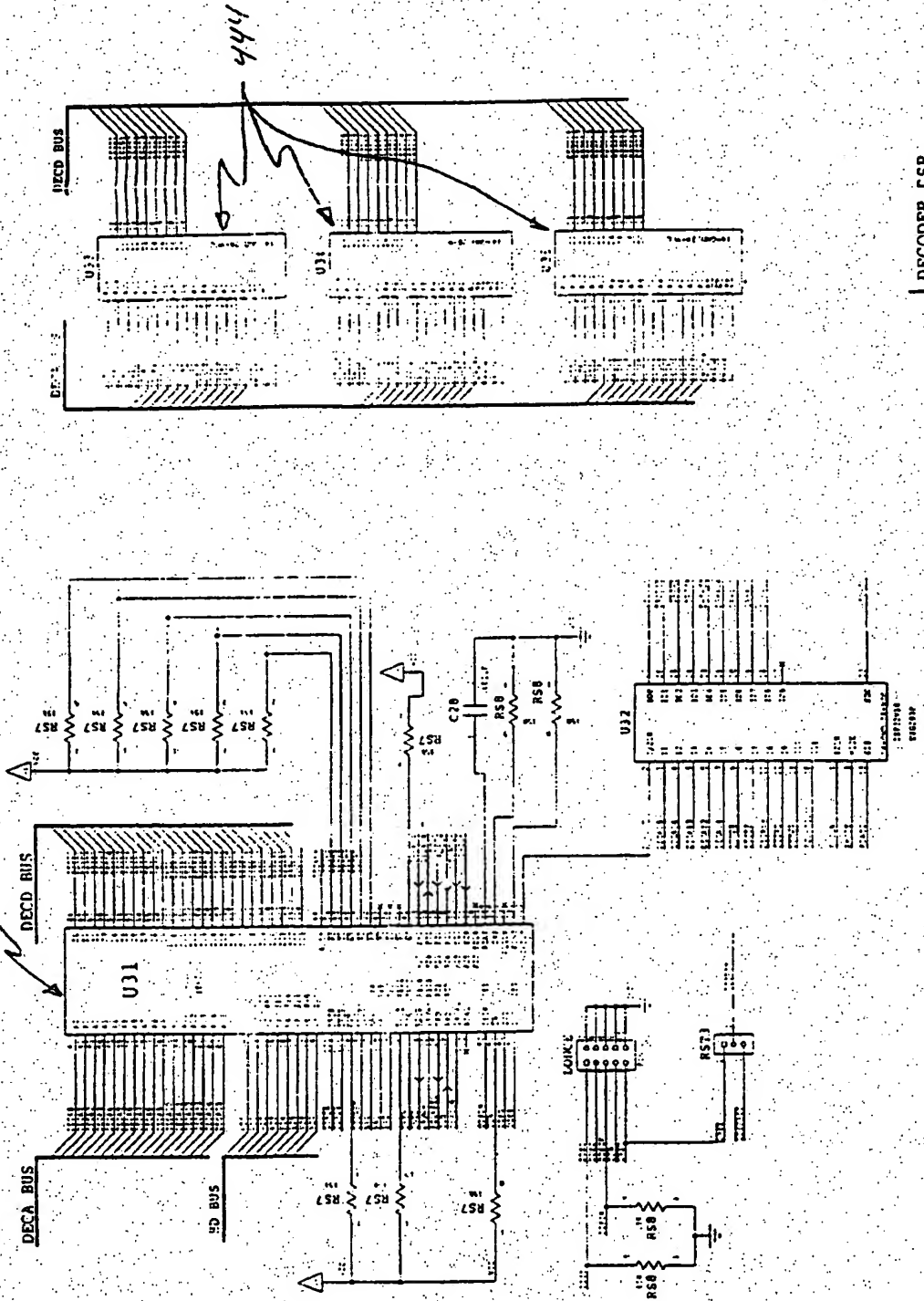


Fig. 16

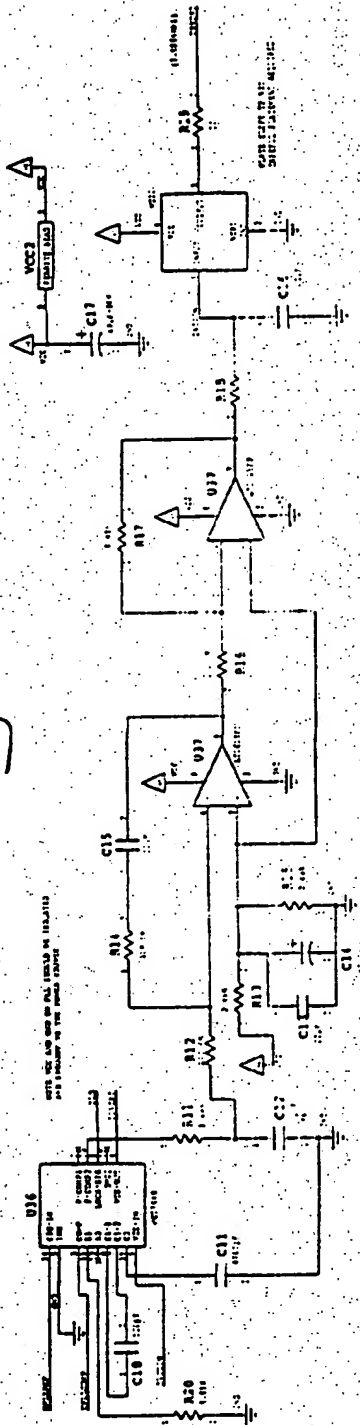
Fig 17



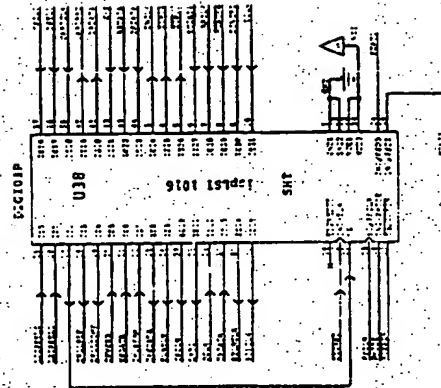
DECODER CSP

BAD ORIGINAL

Fig. 18



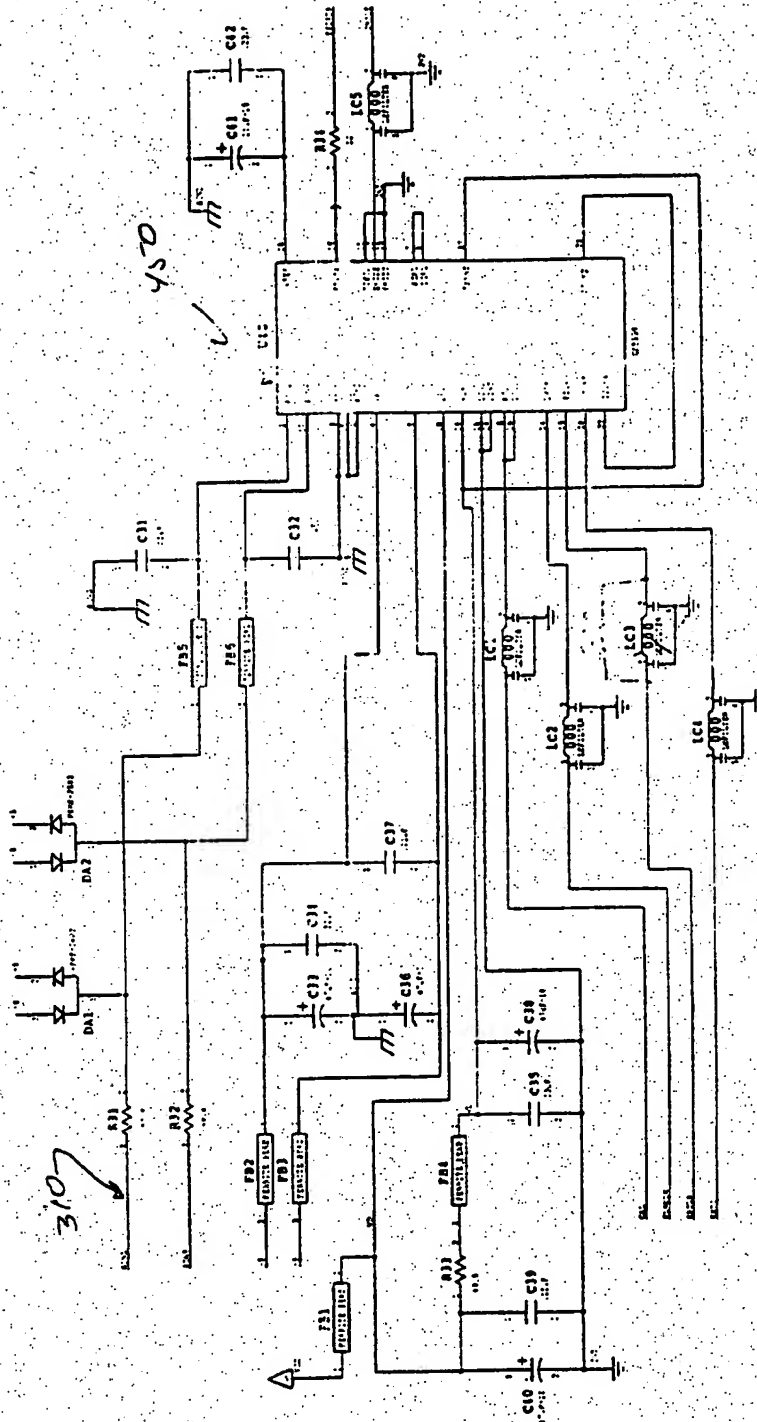
8448



DECODER PLL

19/20

Fig. 19

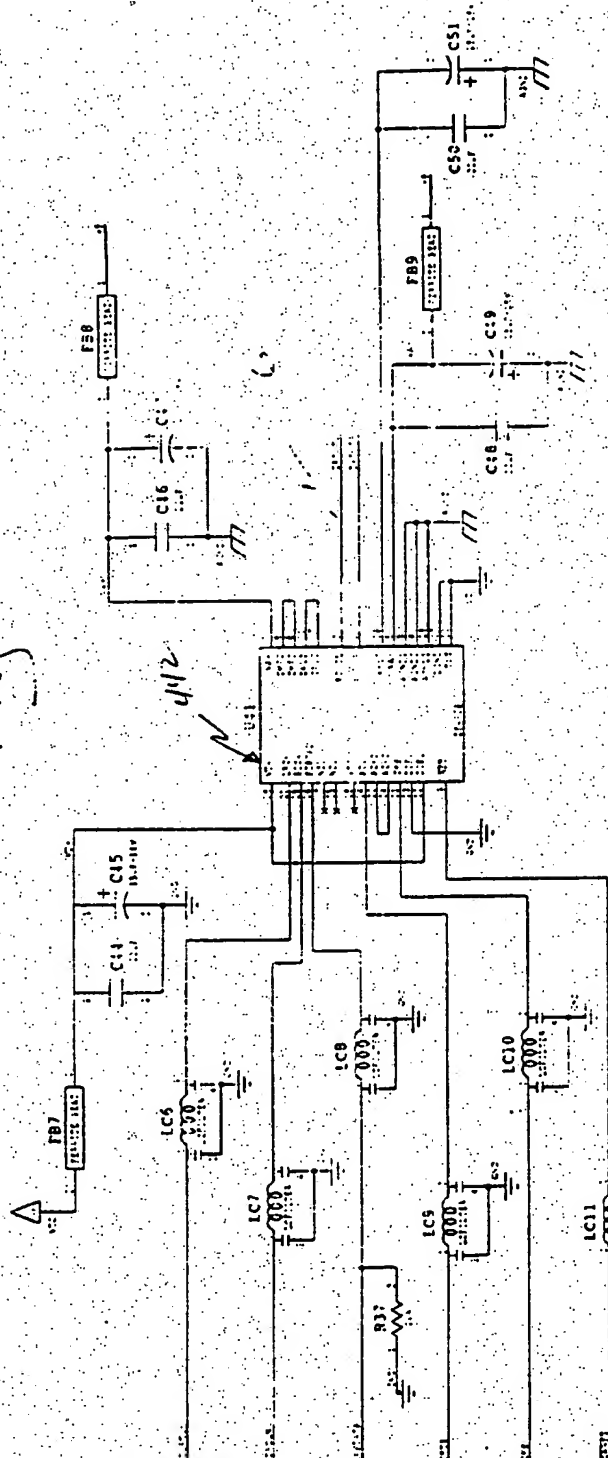


ADC STAGE

BAD ORIGINAL



Fig. 20



DAC STAGE

BAD ORIGINAL



INTERNATIONAL SEARCH REPORT

International application No.

PCT/US96/04835

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04M 11/00

US CL :379/93

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 379/93, 90, 98, 101

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and; where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US, A, 5,325,423 (LEWIS) 28 JUNE 1994, col. 1, lines 31-44, 49-51; col. 8, lines 52-64; col. 9, lines 5-68.	1

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	*T	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
A document defining the general state of the art which is not considered to be part of particular relevance	*X*	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
E earlier document published on or after the international filing date	*Y*	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
L document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reasons (as specified)	*Z*	document member of the same patent family
O document referring to an oral disclosure, use, exhibition or other means		
P document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search

03 JULY 1996

Date of mailing of the international search report

24 JUL 1996

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231

Facsimile No. (703) 305-3230

Authorized officer

Stella Woo

Telephone No. (703) 305-4395

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.